TRADITIONAL AND DEEP LEARNING APPROACHES TO COLOR IMAGE COMPRESSION AND PATTERN RECOGNITION PROBLEMS

Lorenzo E. Jaques

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2021

APPROVED:

Colleen P Bailey, Major Professor
Murali Varanasi, Committee Member
Parthasartathy Guturu, Committee Member
Kamesh Namuduri, Committee Member
Shengli Fu, Chair of the Department of Electrical Engineering
Hanchen Huang, Dean of the College of Engineering
Victor Prybutok, Dean of the Toulouse Graduate School

Jaques, Lorenzo E. *Traditional and Deep Learning Approaches to Color Image Compression and Pattern Recognition Problems*. Master of Science (Electrical Engineering), August 2021, 85 pp., 9 tables, 25 figures, 1 appendix, 39 numbered references.

This thesis includes three separate research projects focusing on computer vision principles and deep learning pattern recognition problems. Chapter 3 entails color quantization applications using traditional Kmeans clustering techniques and random selection of color techniques within the red, green, blue (RGB) color space to maintain a high-quality image while significantly reducing image file size. Chapter 4 consists of a handwriting character recognition algorithm using backpropagation to classify 70,000 handwritten values from US Census Bureau employees and high school students. Chapter 5 proposes a novel classification technique for 109,446 unique heartbeat samples to identify areas of interest and assist medical professionals in diagnosing heart problems.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for providing me the tenacity never to give up, the mental capacity to understand and dissect a wide range of technical subject content, the ability to live off of minimal sleep, and the capability of being able to write a thesis successfully.

Mama and Pops, I can not express how proud I am to be your son, and I appreciate you both supporting me through thick and thin. Mama, let's go find the moon! Pops, I did it! Kristina, thank you for always inspiring me to be the best version of myself and push me to live my best life. Eliana, Thank you for continually gassing me up and being my hype man and giving me the confidence to know I was doing the right thing. To my inner circle of friends, Joseph Ryan Adams, Michael James Kuehn, Brian Arthur Smoot, and Casey Joshua Spoon, I would like to extend my sincerest thanks to you for everything. A special shout out to my doggo, Vato, for being my late-night study buddy and my cat, Ash, for always camping out on my desk, which was sometimes in the most inconvenient location. Thank you to the rest of my family and friends for their unconditional love and constant support.

Thank you to my committee Prof. Colleen Bailey, IEEE Life Fellow and Founding Electrical Engineering Chair, Dr. Murali Varanasi, Prof. Parthasarathy Guturu, Prof. Kamesh Namuduri for all of the help in getting me here. Thank you to Arthur Depoian II and the OSCAR LAB, Dong Xie, Shengjun (Daniel) Zhang, Eric King, Ethan Murrell, Garrett Cayce, Hae Jin (Hayley) Kim, Miguel Rivera, Nathaniel Collins, Nicholas Chiapputo, and Zachary Walker. Thank you to the IEEE UNT Student Branch, the UNT EE Professors, and the UNT EE Admin Staff.

Last but not least, I would like to thank my amazingly wonderful wife, Abigail. You are remarkable in every way possible, and I appreciate your understanding of the sacrifices needed to achieve my academic goals. You are my rock and a real blessing, and I am eternally grateful for your unwavering support while I went through my educational journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

What is considered intelligence? When will machines achieve the ability to smell, touch, feel, and reason? Will future AI systems share a standard dialect? The best researchers in the world are working on these questions. Specifically, the machine learning world's current focus is to provide a poorly-defined unconstrained set of goals by turning any problem into a set of numbers and applying complex formula manipulations. The best approach presently is modifying the parameters of machine learning with optimization tuning to correct any problem. AI is inching closer to obtaining human senses' fundamentals with the advancement in CPU and GPU hardware continuously evolving. AI systems are starting to achieve a well-defined, specifically formally defined, finite set of goals using unsupervised learning.

The remainder this thesis is organized as follows; Chapter 2 provides the machine learning concepts needed to complete the included research projects; Chapter 3 entails computer vision applications using color quantization concepts to compress images from the USC-SIPI image database; Chapter 4 consists of implementing a two-layered artificial neural network using the MNIST handwritten database. Chapter 5 implements a convolutional neural network using the MIT-BIH heartbeat database. Chapter 6 is the conclusion of the thesis.

CHAPTER 2

BACKGROUND

This chapter describes the machine learning concepts necessary to complete the research works presented in this thesis including data handling, artificial neural networks, convolutional neural networks, and validation metrics commonly used to verify the efficacy of implemented machine learning algorithms.

2.1. NumPy Array

Numerical Python (NumPy) [2] is the preferred tool utilized for anything involving data science or machine learning and can be considered a MATLAB replacement. A lot of the commonly used implementation for reading in lists like Pandas library, store images and is the back end for the machine learning applications with the utilization of tensor libraries. It is far superior to using a python list or a python loop. NumPy arrays store data in multidimensional arrays. NumPy uses "fixed type," which is more efficient for storing and manipulating data. For example, the computer does not see a "5" but, the binary representation of 8-bits (one byte) of an integer "5" being "00000101". The NumPy function converts "5" into an Int32 of four bytes of data memory space with the leading three bytes padded with zeros. However, with lists, there is a substantial amount more information that is needed for the same integer "5". Lists use a specified built-in int type for python of four categories: Object Value, Object Type, Reference Count, and Size.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ \mathbf{5} & 6 & 7 & 8 & 9 \end{bmatrix} \tag{1}$$

The object value is the data from which the object contains its specific bits. The object types can be booleans, integers, long integers, floating-point numbers, and complex numbers. The object count is how many times the integer is pointed to in the data structure and the size of the specified integer value. As seen in Figure 2.1, a lot more space is required

for lists. Due to this reason, NumPy allows for fewer bytes of memory needed to be read by the computer. When iterating through a NumPy array, there is no type checking when reading through objects, but there is a type check when reading through lists. The lists require a type check for an integer, float, string, or boolean and checks each element and what type it is.

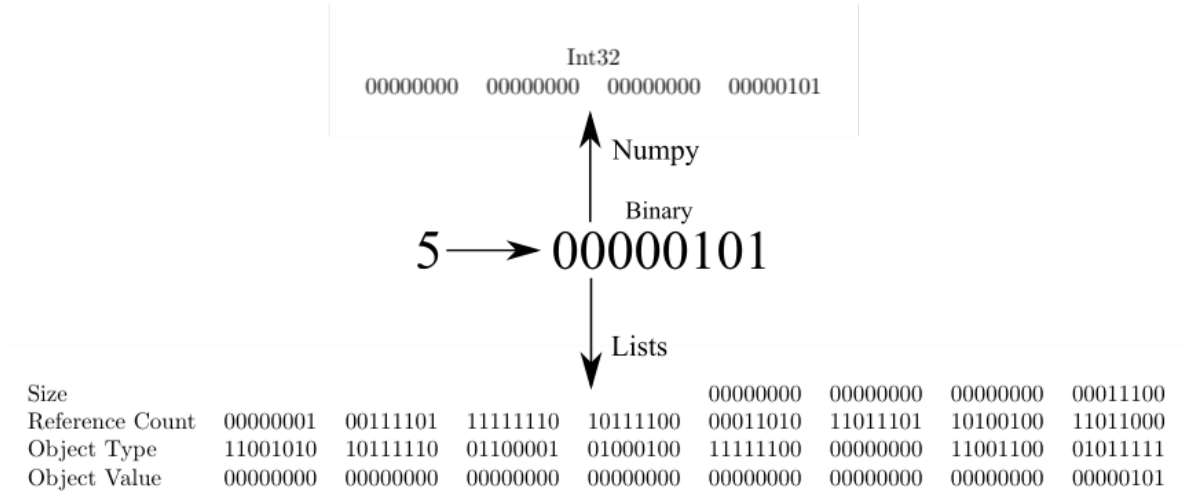


FIGURE 2.1. Computer Reading Lists vs. Numpy

The list's data is scattered around in the eight memory blocks seen in Figure 2.2, and those blocks are not next to each other. The list array contains pointers to the information scattered throughout the computer memory. The computer has to bounce around the memory to find the areas of importance to perform functions on the set information. Numpy utilizes contiguous memory, which is considered an unbroken block of memory. All eight blocks are next to each other, with a vital location of the memory's start with the total size and memory block type.

When the memory is aligned next to each other, the CPU can utilize single instruction multiple data (SIMD) vector processing [3]. The SIMD vector unit can add on values simultaneously instead of completing the computational task one addition at a time when the data is spread amongst multiple memory blocks. It allows for effectively using the cache for quicker memory in the computer for easier access. In the list case, the computer can only load a portion of information before being required a reload into the cache, creating longer

memory lookups within the computer.

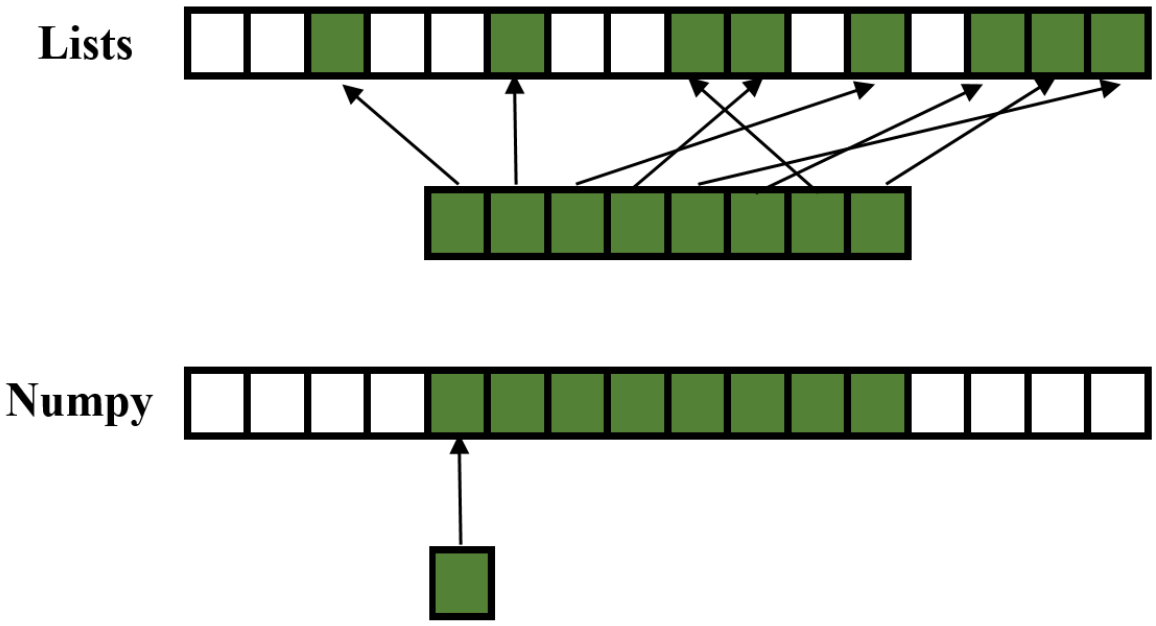


FIGURE 2.2. List vs. Numpy Array Storage

The last and final benefit of using NumPy over Lists is the computation syntax ease of use. As seen in Table 2.1, simple computation in List is not allowed, and more discretization inputs are required. The requirements for additional lines of code adds up when more complex processes are required.

TABLE 2.1. Computations Using Lists versus NumPy Arrays

	List	NumPy Array
Python Input	A = [1, 2, 3]	A = np.array([1, 2, 3])
	B = [1, 2, 3]	B = np.array([1, 2, 3])
Console Output	A * B = ERROR	A * B = ([1, 4, 9])

2.2. CPU vs GPU Computation - Tensorflow

CPUs handle multiple tasks and do many things simultaneously like calculation, memory fetching, IO, and interrupts. It has a large complex instruction set running serially to process items in a specific order with a set delivery schedule. CPUs are superior in scalar computations but lack speed with matrix computations.

Graphics Processing Units (GPU) are designed to complete one task on computers. That task is for lots of math and run instruction sets in parallel while getting things done simultaneously, moving fast from A to B. The focused design is why they are crucial for video games. Every visualization on a video game requires fast mathematical computations to render every blade of foliage, the dynamic change in lighting, and the frame's movement, which involves many parallel computations at once. The most common usage has become training machines to achieve some intelligence level due to how evolved GPUs have become. Deep neural networks require multiple data points to be processed simultaneously for training. GPUs are the physical foundation for artificial intelligence due to this very reason.

GPUs physically have a larger memory bandwidth, use parallelization, and have more memory access. Specifically, the GPU can fetch much larger amounts of data from the RAM to place in the GPU's memory. Simultaneously, the GPU processor remains idle allowing multiple nodes of transferring information where the GPU processors can always access data for continuous calculations which is called parallelization. Coupling parallelization with the larger memory bandwidth reduces the time a GPU would be waiting. GPU's CACHES are smaller in size than CPU, which allows for much faster access, and the GPU's streamlined processors with up to 1000 more times registers than CPUs. All of the memory enhancements together on a GPU are faster in matrix operations. Faster processing time in matrix multiplication allows for rapid operations in deep learning frameworks. The CUDA toolkit [4, 5] provides an API that enables access GPU components like streamlines, caches, and registers. Deep learning frameworks like Tensorflow allow the processing of complex matrix multiplication, which is one of the fundamental operations in Neural Networks [6].

2.3. Data Organization

Exploratory Data Analysis (EDA) [7] needs to be completed before creating a machine learning model. It is great to start working and building everything out, but how does one know if the dataset attempting to have machine learning algorithms is valid. It is necessary to plan the best approach to know if the dataset fits one model better. This step can hugely influence the results obtained after the machine learning process underway and save

an excessive amount of time. The questions which need consideration when exploring a data set are:

- Is there anything missing from the data?
- What kind of data do you have?
- Where does the data get parsed to feed into the network?
- How do you ensure the data is valid?

To overcome most of these obstacles with data in my research, I looked for open-source datasets in various research applications and implemented my algorithm design. Some of the datasets have a associated data dictionary, and best practices are to document those selected features. The distribution of the data can help show the visual validity of the data. Depending on the source, feature imputation is needed to fill in missing values. Regardless of the machine learning algorithm implemented, feature encoding must turn all datasets into numerical values for the algorithm to execute. I ran into problems with the "curse of dimensionality," which refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings, such as the three-dimensional physical space of everyday experience [8]. The last obstacle encountered in my research was dataset imbalances. Sometimes the data needs to be reshaped for your learning model to function correctly. Further discussion regarding imbalanced datasets will be found in Chapter 5.1.

2.4. Neural Networks

A neuron is the computational building block for the human brain and has 100-1000 trillion synapses. An artificial neuron is the computational building block for the neural network (NN), which has 1-10 billion synapses. By comparison, human brains have 10,000 times the computational power of computer brains. A NN is predefined with a fixed number of nodes, and layers. Human brain neurons die and are born all the time. Supervised learning NN are exceptional at memorization but poor at reasoning. NN are considered Deep learning, which is a subset of machine learning. They are commonly referred to as multilayer

perceptron (MLP) and convolutional neural networks (CNN). The primary purpose of a NN is to build a multilayer resolution of the data. The majority of mathematics used in NN is matrix calculus which works are described Parr and Howard [9].

NNs have several layers where each layer forms a higher-order representation of the input. NNs are composed of simple elements operating in parallel. Biological nervous systems inspire these elements. As in nature, the network function is determined by the connections between elements. We can train a NN to perform a particular function by adjusting the connections (weights) between elements. NNs are not efficient learning machines. Where humans can learn from one example, NNs require extensive data and are inefficient at learning from data. NNs are generally considered supervised learning because the network demands the manual selection of the ground truth labeling of the data for the network structure. This process can be computationally costly to annotate real-world data. Getting NNs to perform well for large-scale datasets requires a lot of hyperparameter tuning. In the end, humans treat a NN like a "black-box" because of the high dimensionality of matrix calculations that can be computed by a NN in a short amount of time instead of being calculated by hand. NNs require a designation of hyperparameter modifications to achieve reasonable results by tuning the training, learning rate, loss function, mini-batch size, number of iterations, momentum gradient smoothing, and an optimizer selection. The global operations of a NN involve convolution, pooling, activation function, and backpropagation.

Figure 2.3 is a visual representation of a single artificial neuron. The neuron takes a set of weighted inputs ($x_n \cdot w_n$) and sums them together. A bias value b is applied to each neuron before an activation function that takes the sum plus the bias and compresses it together to produce a 0-1 output signal for classification. Two of the most common activation functions are the rectified linear unit (ReLU) with corresponding Equations (2) and (3) and the sigmoid function with corresponding Equations (4) and (5), illustrated in Figure 2.5. If the output does not match the ground truth or the expected output, then the weights are punished accordingly. The recursive process continues until the perception does not make any more mistakes.

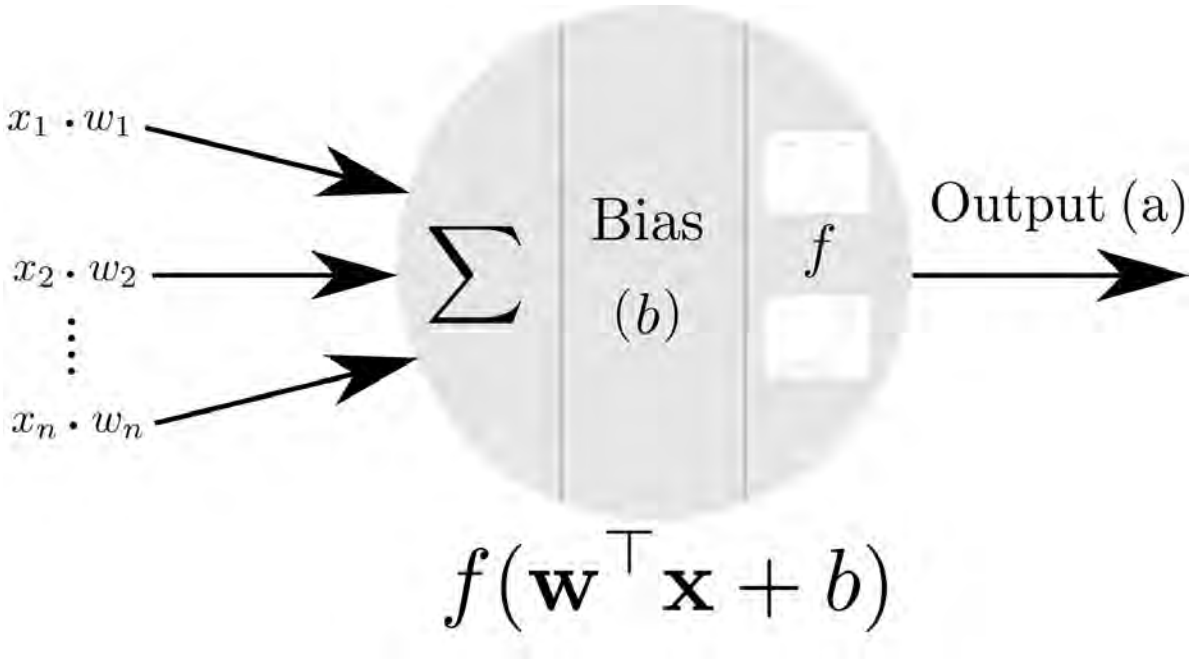


FIGURE 2.3. Neuron

$$R(z) = \begin{cases} z: & z > 0 \\ 0: & z <= 0 \end{cases} \tag{2}$$

$$R'(z) = \begin{cases} 1: & z > 0 \\ 0: & z < 0 \end{cases} \tag{3}$$

$$s(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

$$s'(z) = s(z) \cdot (1 - s(z)) \tag{5}$$

To further expand, Figure 2.4 is a fully connected simple NN with three nodes at the input layer, five neurons at the hidden layer, and two nodes at the output layer. Backprop-

agation and gradient descent is the apparatus of machine intelligence. Backpropagating the loss function through the gradients in each local layer is how a NN learns a model. The idea of backpropagation was introduced in 1960 by Henry J. Kelly [10] and was validated through extensive experimentation in 1986 by Rumelhart et al. [11] and has been one of the most studied and implemented algorithms for NN learning ever since. A forward pass computes the network output at every neuron, and finally, the output layer also calculates the "error" between a and b.

A backward pass computes the gradients, so instead of one on the outputs, it will be the error to backpropagation to use chain rule from output layer to previous layers. Once the gradient is known, the weights are updated in the opposite direction of the gradient. The amount the adjustment is made for the loss to decrease is called the learning rate. The learning rate can be the same across the entire network, or it can be different at every individual weight, depending on the model's experiment and parameters. Depending on the dataset injected at the input layer, a dropout layer can be added to randomly remove nodes in the network along with incoming and outgoing edges.

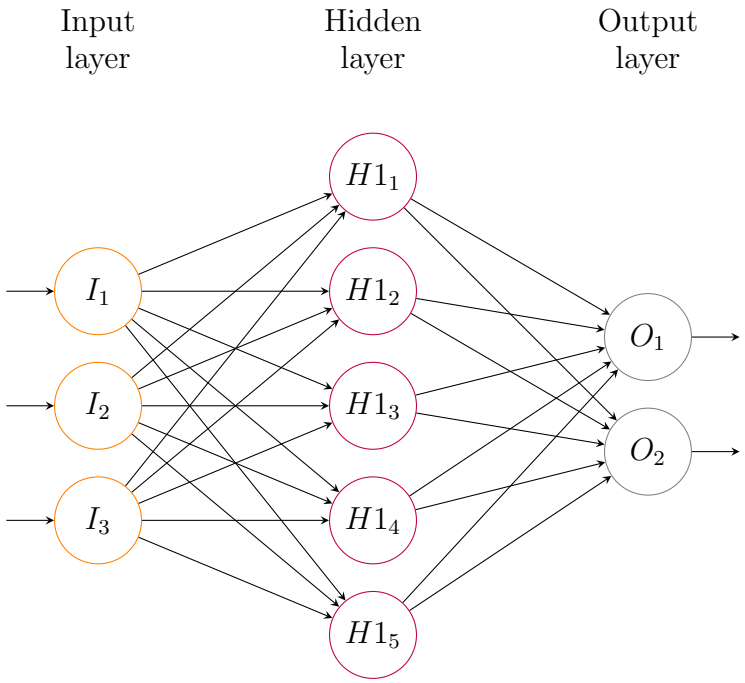
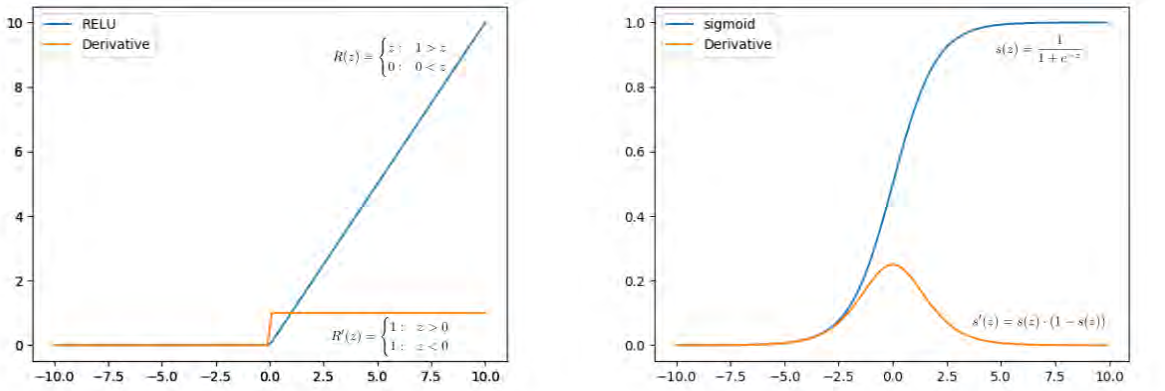


FIGURE 2.4. Simple Neural Network

Machine learning can be considered an optimization theory application where the task is to minimize an objective function. The weights and bias are the decision variables for which NNs have an excessive amount. Given the structure of all NNs with more than one hidden layer, the objective function is nonconvex. The goal of training is to have find the global minima, however, because of nonconvexity, NNs sometimes are stuck in a local minimum or a saddle point. The value of cost function will ideally equal 0 in the training session. The issue which makes optimization hard is the vanishing exploding gradients problem. The gradient depends on how much the weights need to be adjusted. As seen in Figure 2.5b, the sigmoid function derivative is 0, at the tails. When the input to the sigmoid function is exceptionally high or exceptionally low, that derivative will be close to 0. Therefore, the gradient calculated will be 0 negating the backpropagation calculation through the neuron where no learning is happening. The ReLU function derivative illustrated in Figure 2.5a does not have tails, so a whole section of the network could have exploded ReLU gradients due to an error in the initialization process. Two of the most common algorithms used for optimization in NNs are Stochastic Gradient Descent (SGD) [12, 13], and Adam [14]. The Adam algorithm combines the strengths from the root mean square propagation and adaptive gradient descent algorithm. The Adam optimizer algorithm varies the learning rate, diversifying how much each node in the network will change its weights for their activation function versus keeping the learning rates fixed than the standard gradient descent algorithm.

2.5. Convolutional Neural Network

Convolutional Neural Network (CNN) contains the exact characteristics of NNs but have convolving layers that captures the data's spatial features, including sequential datasets. Due to this characteristic, image analysis is one of the most common applications for CNNs because of the resilience and capability in pattern recognition and requiring little pre-processing [15]. Computer vision is challenging since the illumination variability of the same object with drastic lighting differences. For the depth of any image, CNNs can understand the location of an object within the set image by sharing parameters creating a feature map with filters distributed to various parts of the image [16].



(A) ReLU (B) Sigmoid

FIGURE 2.5. Activation Functions

2.6. Validation Metrics for Machine Learning

The most commonly used equation metrics to verify machine learning algorithms are accuracy (6), precision (7), recall (8), and F1 score (9). True positives and true negatives are correctly identified, while false positive and false negatives are not correctly identified.

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative} \tag{6}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{7}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{8}$$

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{9}$$

Validation loss curves and validation accuracy curves are referred to as "hockey stick

graphs" which, in ideal conditions, visually display the performance of the network with a smooth trend curve of the data.

The Structural Similarity Index Measure (SSIM) is used to compare the original data with the mutated data by quantifying the perceptual difference between two grayscaled images using luminance, contrast, and structure [17].

CHAPTER 3

COMPUTER VISION - APPLICATIONS OF COLOR QUANTIZATION

Stuart Lloyd from Bell Labs created an algorithm technique for pulse-code modulation, which was mainly used for scalar quantization in 1957 and was not officially published until 1982 [18] where the derivation of Equation (10) is expressed.

$$f = \sum_{j \in [k]} \sum_{i \in S_j} ||x_i - \mu_j||^2 \tag{10}$$

This algorithm evolved and was coined as "k-means" courtesy of MacQueen et al. in 1967 [19]. Kmeans is considered an unsupervised machine learning technique where the model discovers its information and pattern recognition previously unknown. Various Kmeans applications include methods for data clustering [20], image segmentation [21], abnormality detection in extensive data [22], automatic data structured discovery [23]. In the case of this research project, color quantization using Kmeans along with random selection.

In this research, the goal is to maintain the highest quality of the original image and compare the best rate of compression using the strength of the Kmeans clustering algorithm versus a random selection of colors per the image color palette. The dataset chosen for this research is five widely utilized images for computer vision research in the open-source USC-SIPI Image Database [24] as seen in Figure 3.1. The details of the selected images for the USC-SIPI Image database are illustrated in Table 3.1. Only the Mandrill picture results will be displayed in this chapter. The remainder of the image results can be found in Appendix A.

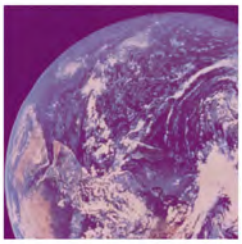
Full colored images are generally represented in the red green blue (RGB) 3-dimensional color space, with each color channel assigned one byte. Therefore, three bytes are required to encapsulate the RGB spectrum or 2^{24}, which equates to 16,777,216 potential color combinations in one colored image. This is why a majority of photos are often referred to as "24-bit images". Minimizing the number of specific colors in an image is called color quan-

TABLE 3.1. USC-SIPI Image Information (24 bits/pixel - png)

Image	Photo Size	Storage Size	Unique Colors
Mandrill (a.k.a. Baboon)	512x512	392.8 kB	230427
Airplane (F-16)	512x512	283.9 kB	77041
Earth from space	512x512	324.7 kB	96901
Sailboat on lake	512x512	347.2 kB	168459
Tree	256x256	189.3 kB	44380

(A) Mandrill

(B) Airplane



(C) Earth

(D) Sailboat

(E) Tree

FIGURE 3.1. USC-SIPI Image Database Selected Images - Mandrill, Airplane (F-16), Earth from space, Sailboat on lake, & Tree

tization. The main focus of quantization of colors is to reduce the number of unique colors while maintaining the highest quality image output [25]. The comparison for Kmeans and random selection will be completed in the RGB color space to illustrate data storage size for the resulting image's compression. The five pictures tested have range of color diversity, so the results exemplify the implemented algorithm's efficacy.

3.1. ꟻLIP - NVIDIA Evaluator for Alternating Images

In the summer of 2020, the NVIDIA Research and Development Team released their full-reference image difference algorithm metric called ꟻLIP [26]. Unlike SSIM, the ꟻLIP algorithm can take in all color channels from the RGB color space and map out the differences

when flipping between two images back and forth without blanking between them. The resulting output image of ꟻLIP shows the error of the visual differences between a reference image compared to a mutated image, which in this research is the Kmeans image. The unique difference from other comparison difference algorithms is to take into account the contrast perceived by humans when alternating between two images. SSIM is designed to operate on grayscale images and find the differences between the two images and does not consider color but the contour difference between images for the structural similarity. The NVIDIA Research and Development Team provided the open-source code for the ꟻLIP algorithm. The ꟻLIP algorithm was implemented in this research for experimental purposes, with interesting results showing the NVIDIA Development Team's works on the set of images used in this research project.

3.2. Implementation

Lloyd's algorithm [18] in combination with the Expectation-Maximization (EM), which process is explained in Bilmes et al. [27], was implemented on the five test photos, and the method is illustrated in Algorithm 1. Once the values are loaded in from the image, the color palette is stored in a numpy array. The input for Kmeans is based on the amount of desired clusters. Values of 2^n were strategically selected to show the full range of the possible RGB color space with the implemented Kmeans and random selection cluster photos.

Algorithm 1 Color Minimization Algorithm Implementation

```
1: procedure LOAD IMAGE DATA
2:     Convert floats into 8 bits integer
3:     Transform into 2D numpy array
4:     Designate desired k clusters
5:     Initialize k centroids at random state
6:     while centroids positions are non stationary do
7:         Expectation: assign each pixel to closed centroid value
8:         Maximization: calculate new centroid mean of each cluster
9:     end while
10:    Reshape image with desired k clusters
11: end procedure
```

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

(F) 64-color clusters

(G) 128-color clusters

(H) 256-color clusters

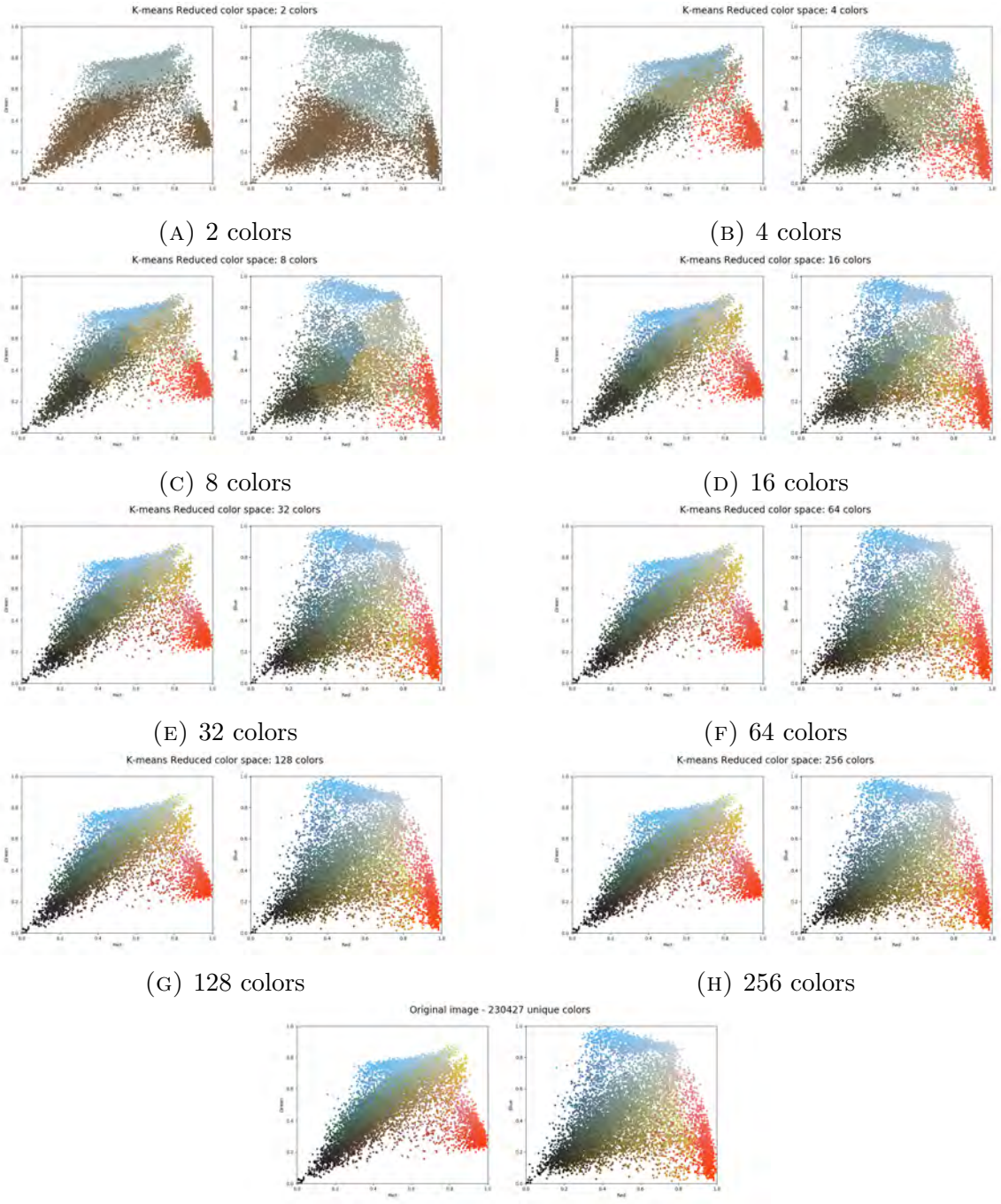
(I) Original Image

FIGURE 3.2. Mandrill - Various Kmeans Color Clusters Applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

3.3. Results - Kmeans - Mandrill

Figure 3.2 is the output of images ranging from 2^1 to 2^8 across the RGB color space with all respected Kmeans outputs of the image. The algorithm considers the mean points in the cluster and bases the result on the "closeness" measured by the Euclidean distance. A majority of the convergence happens in the first few iterations of the EM calculation for the clusters most representative of the image. Figure 3.3 exhibits the behavior of the various color clusters corresponding to the Figure 3.2 applied via the input of the desired clusters.

Figure 3.4 shows the quantifiable comparison between the Kmeans and how well the



(A) 2 colors (B) 4 colors

(C) 8 colors (D) 16 colors

(E) 32 colors (F) 64 colors

(G) 128 colors (H) 256 colors

(I) 230427 unique colors

FIGURE 3.3. Mandrill - Various Kmeans Color Scatter Plots (2, 4, 8, 16, 32, 64, 128, 256, & 230427)

output image performed at various iterations of 2^n and provides insight on how much each Kmeans image maintains quality when sweeping through the RGB color space by using the SSIM index. An argument can be made that the SSIM can provide the user insight into the

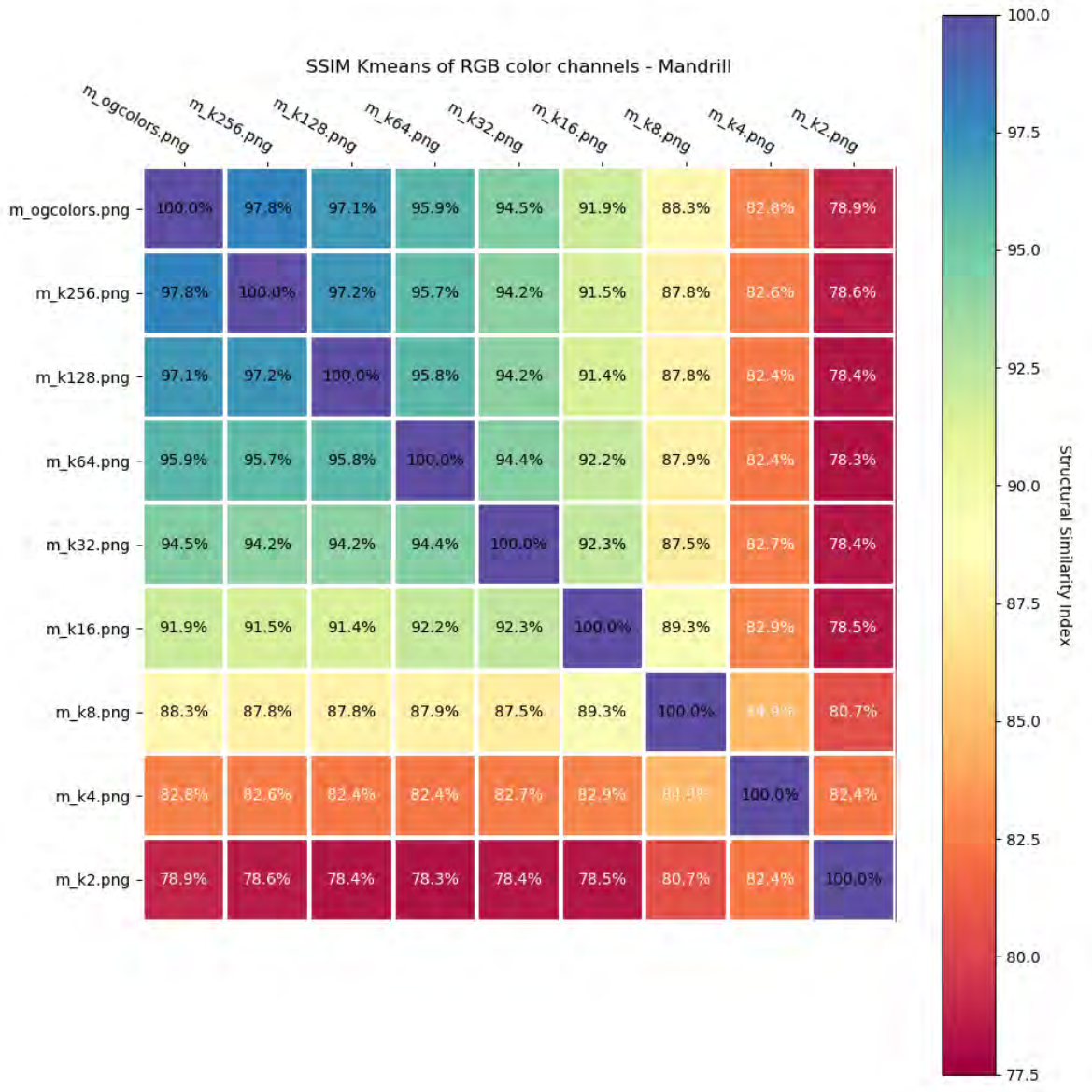


FIGURE 3.4. SSIM Heatmap - Kmeans RGB color channels - Mandrill

optimal value of clusters in the input image. Upon further review of the SSIM heatmap, the first non-linear decrease happens between m_k32 (2^5 a.k.a. 32-color clusters) and m_k16 (2^4 a.k.a. 16-color clusters). When inspecting Figure 3.2e and Figure 3.2d, the image quality starts to break down and is most noticeable in the eyes, which are different colors.

3.4. Results - Random Color Selection - Mandrill

Almost an identical process is followed for the random selection of colors from the RGB color space. After the image is loaded in, three random colors are selected based on

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

(F) 64-color clusters

(G) 128-color clusters

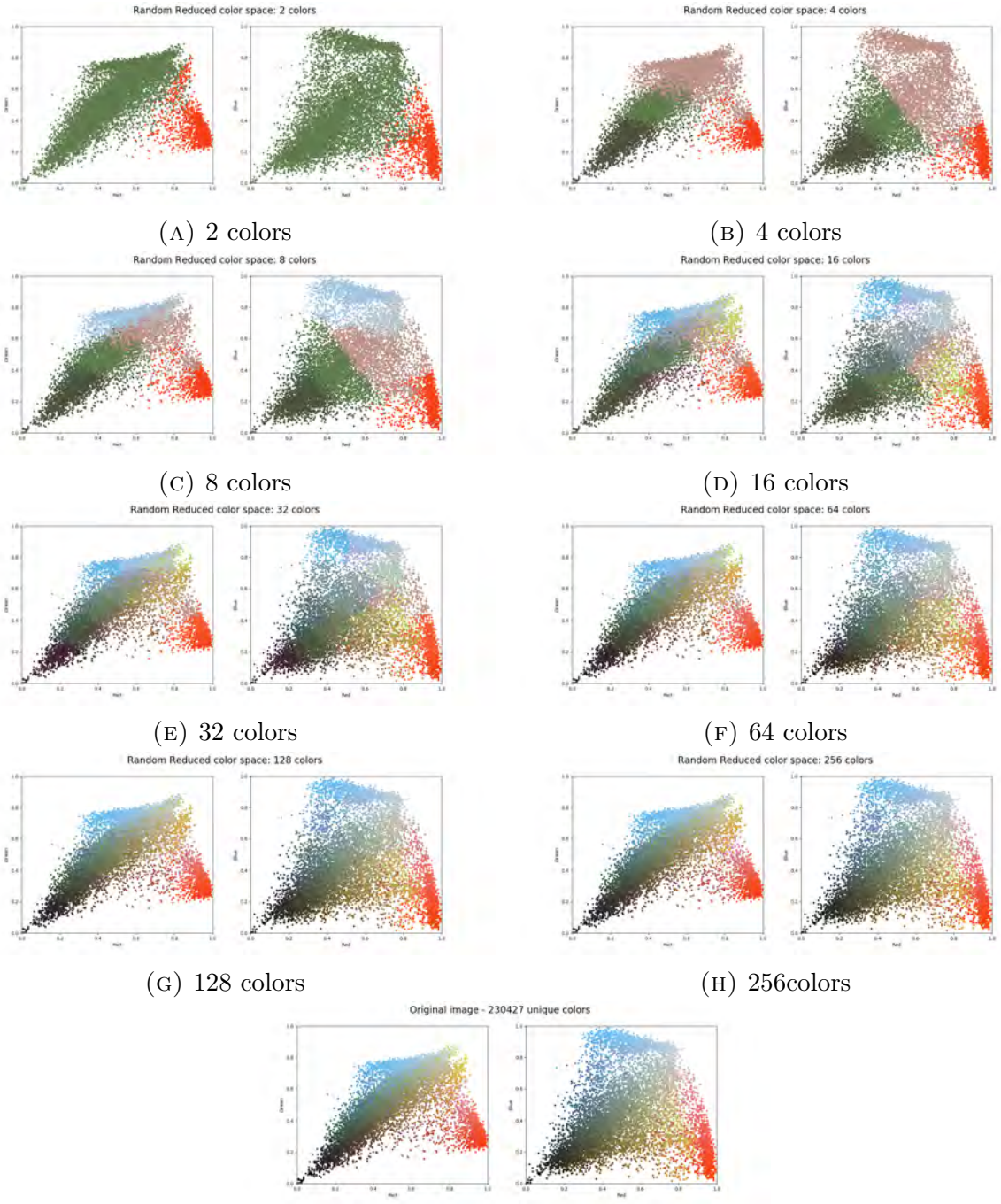
(H) 256-color clusters

(I) Original Image

FIGURE 3.5. Mandrill - Various Random Color Clusters Applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

the numpy array of the image for the palette. The chosen clusters are then remapped to the closest color chosen according to the Euclidean distance to compute the minimum distances between one point and a set of points. In essence, the random color cluster should ideally be mini-representations of the image as a whole, depending on how many are desired. The more clusters were chosen randomly from the color palette of the image, the more accurate representation of the original image will.

To compare equally, the output images were also ranging from 2^1 to 2^8 across the RGB color space with all respected random selection outputs as seen in Figure 3.5. Figure



(A) 2 colors (B) 4 colors

(C) 8 colors (D) 16 colors

(E) 32 colors (F) 64 colors

(G) 128 colors (H) 256colors

(I) 230427 unique colors

FIGURE 3.6. Mandrill - Various Random Color Clusters Applied (2, 4, 8, 16, 32, 64, 128, 256, & 230427)

3.6 exhibits the behavior of the various color clusters corresponding to the Figure 3.5 applied via the input of the desired clusters. The SSIM heatmap displayed in Figure 3.7 illustrates that the random color selection from the color palette does not perform as well as the

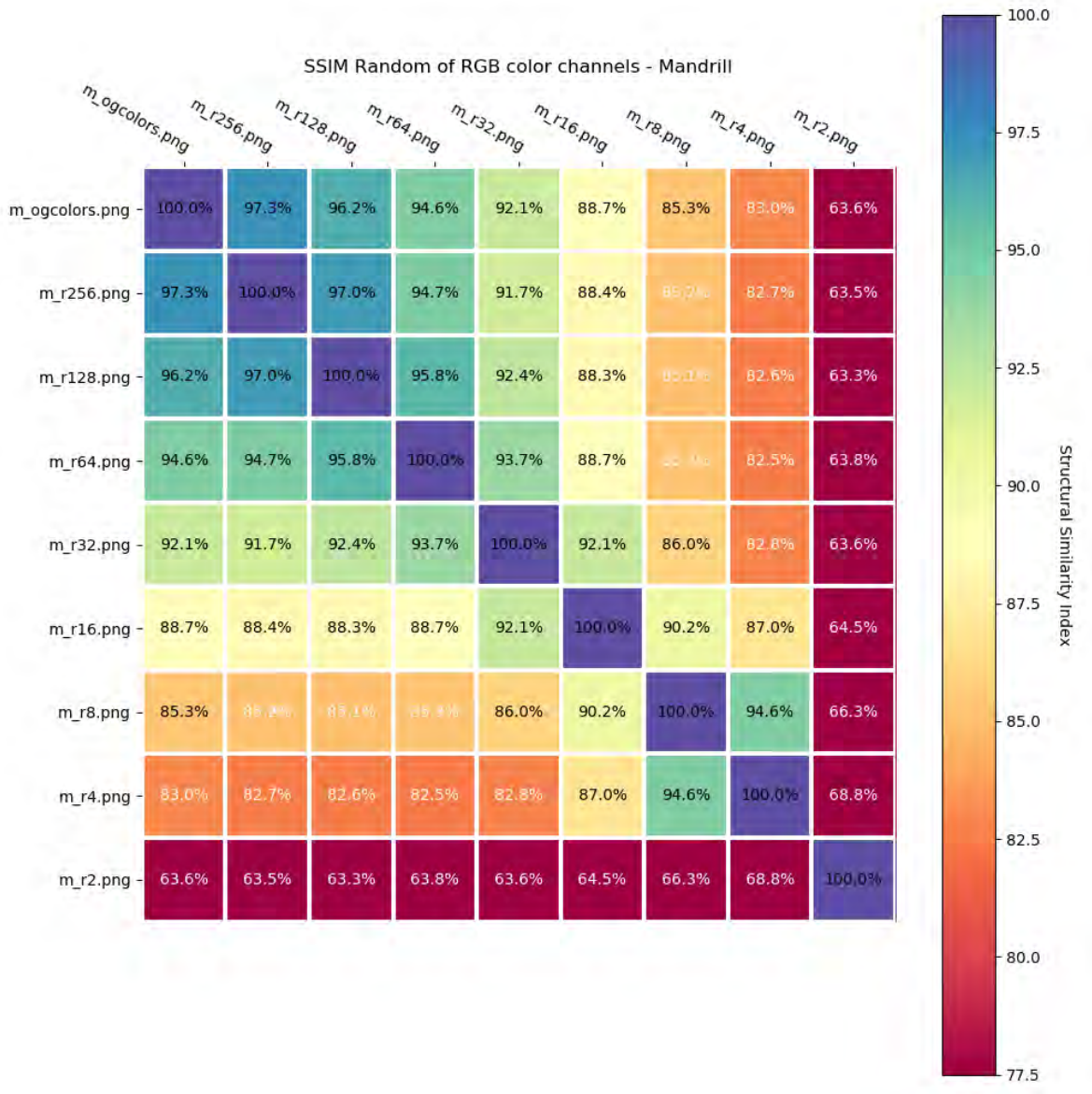
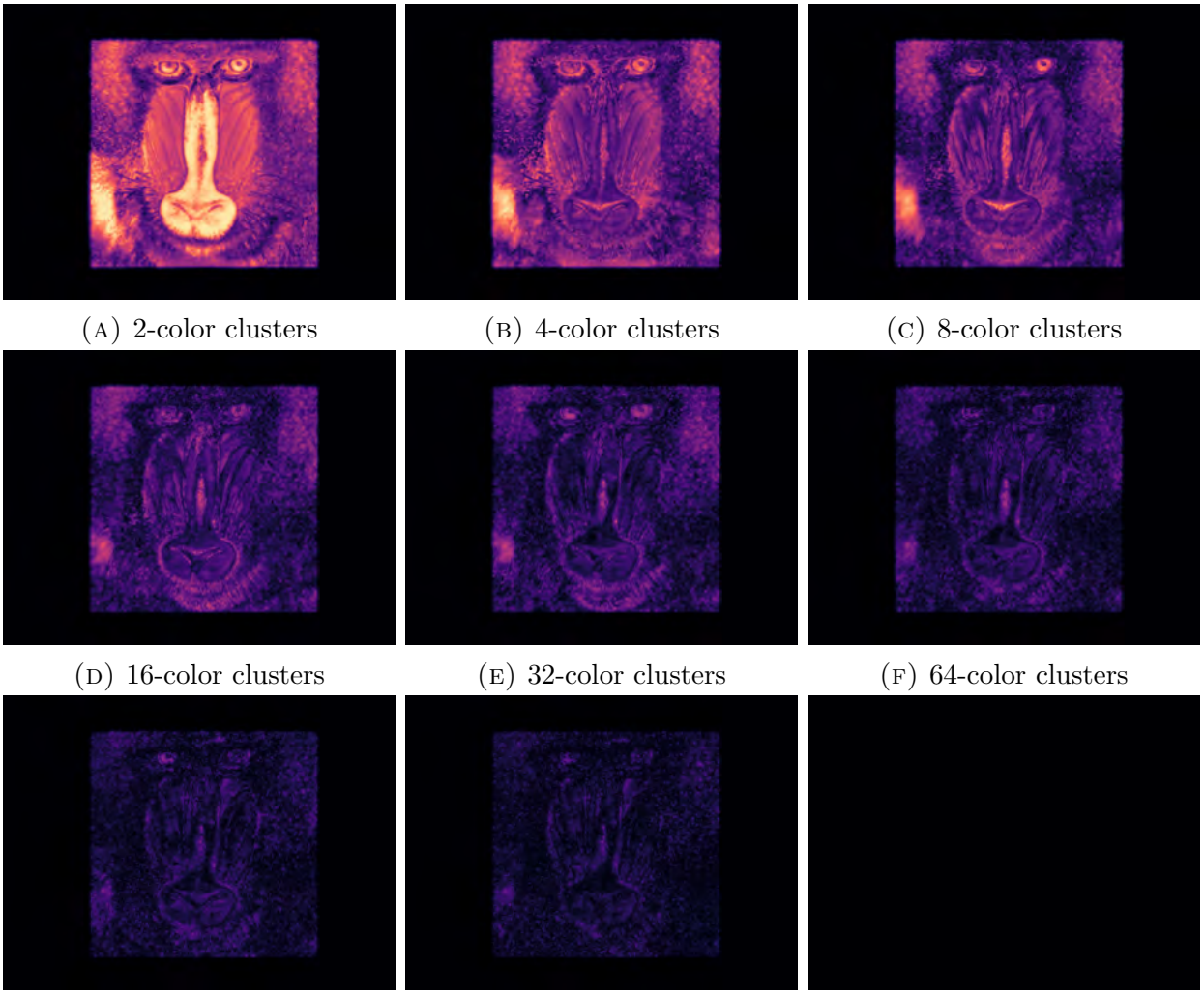


FIGURE 3.7. SSIM Heatmap - Random RGB Color Channels - Mandrill

Kmeans algorithm across the board. The interesting aspect of randomly selecting colors is the examination of the same first non-linear decrease happens between m_r64 (2^6 a.k.a. 64-color clusters) and m_k32 (2^5 a.k.a. 32-color clusters). Even then, the eye colors are not evenly distributed due to the nature of randomly assigning colors versus the more uniform eye color as seen in Kmeans lower cluster of 32 as seen in Figure 3.2e.



(A) 2-color clusters (B) 4-color clusters (C) 8-color clusters

(D) 16-color clusters (E) 32-color clusters (F) 64-color clusters

(G) 128-color clusters (H) 256-color clusters (I) Original Image

FIGURE 3.8. Mandrill - ꟻLIP Kmeans Color Clusters (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

3.5. ꟻLIP Results - Mandrill

As stated in Section 3.1, the NVIDIA Development Team's full-reference image difference algorithm, ꟻLIP, was implemented on the Kmeans images of ranging from 2^1 to 2^8 with the results seen in Figure 3.8. Works proven by the NVIDIA Development Team in Andersson et al. [26], the ꟻLIP algorithm creates error maps that align better-perceived errors between reference and test images. The SSIM was taken of all ꟻLIP Kmeans images to the ꟻLIP image reference, and the results can be seen in Figure 3.9. Since the ꟻLIP metric algorithm is still in its infancy, no documentation was publicly available.

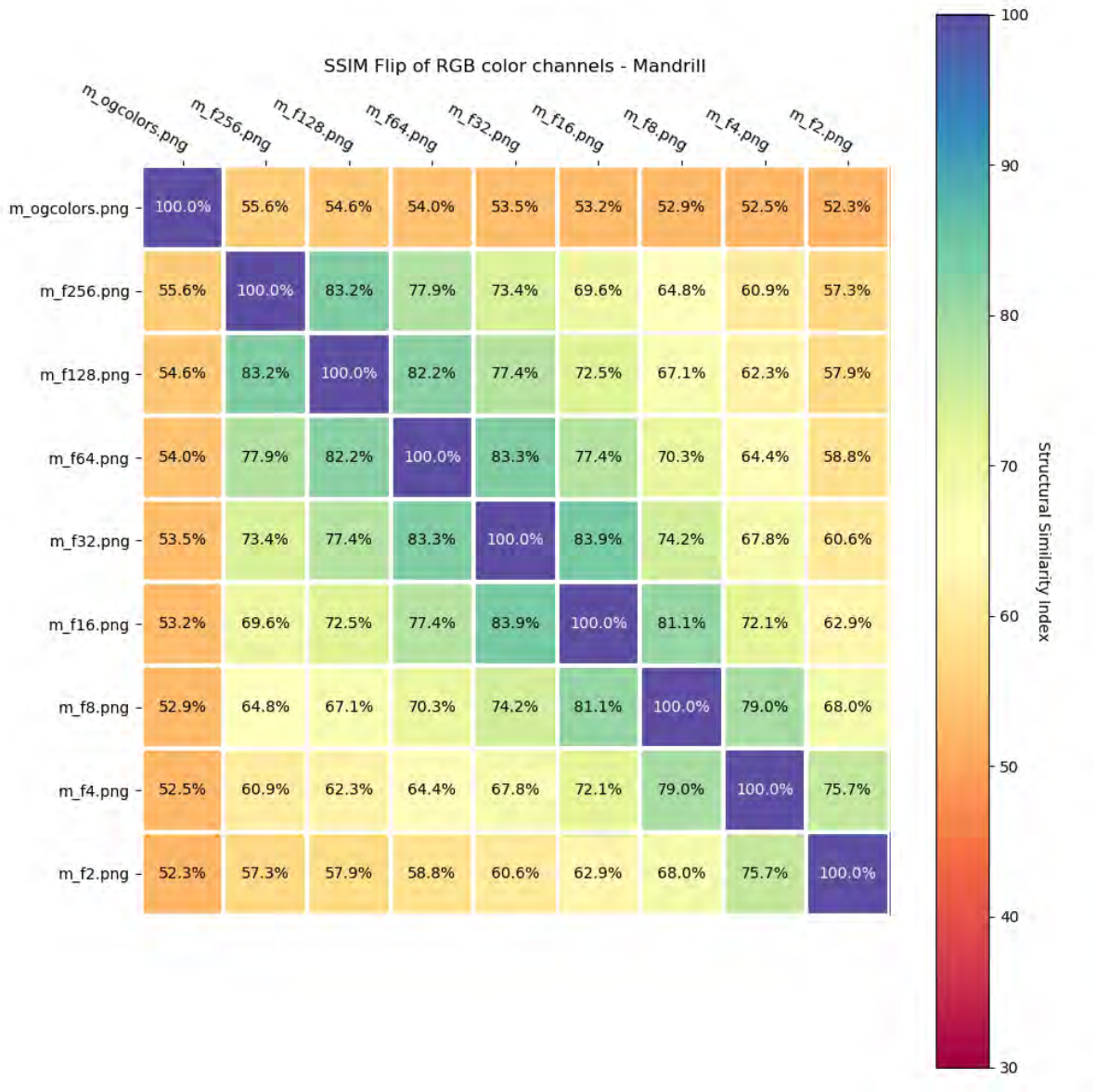


FIGURE 3.9. SSIM Heatmap - ꟻLIP Kmeans RGB Color Channels - Mandrill

3.6. Results - Storage Space - Mandrill

Table 3.2 expresses the physical storage size needed for the mandrill variants utilized in this research project for both the Kmeans and the Random Selection ranging from 2^1 to 2^8. Used in comparison with the SSIM heatmap, the user can decide how many k clusters are desired in unison with the corresponding compression rate.

TABLE 3.2. Mandrill Compression Comparison

	Storage Size	
Cluster Size	Kmeans	Random Selection
Original	392.8 kB	392.8 kB
256	370.0 kB	371.7 kB
128	342.9 kB	347.7 kB
64	301.2 kB	299.7 kB
32	230.3 kB	219.4 kB
16	147.9 kB	132.3 kB
8	91.2 kB	77.6 kB
4	42.3 kB	45.8 kB
2	28.2 kB	7.4 kB

3.7. Conclusion

In this research project, Lloyd's Kmeans algorithm was implemented in parallel with Random Selection of colors in the RGB color space. Extensive testing was completed with comparison to illustrate the data storage size of the resulting images. Based on the results, the Kmeans clustering is the superior algorithm for color quantization by requiring a lower cluster count of colors needed to resemble the original image with minimal loss in quality. An argument can be made that the SSIM can provide the user insight into the optimal value of clusters for the input image paired with the compression results. NVIDIA Research and Development team's new algorithm metric from July 2020, ꟻLIP, was implemented in this project. An SSIM heatmap was created based on the ꟻLIP images of the Kmeans outputs. The remainder of the results from the four images in Figure 3.1 can be found in Appendix A.

CHAPTER 4

TWO-LAYERED ARTIFICIAL NEURAL NETWORK FOR HANDWRITTEN CHARACTER RECOGNITION

The Modified National Institute of Standards and Technology (MNIST) dataset was evolved from the NIST Special Database 19 having an original 20x20 pixel box aspect ratio. The original collection of the training set was from Census Bureau employees, and the testing set was collected from high-school students [28]. Works completed by LeCun et al. [29] illustrated that there was a significant discrepancy from the NIST handwriting samples from adults versus the teenagers. This was because of how clean and easy is was to recognize the adult samples in comparison to the teenagers along with the process in creating the new database. The MNIST handwritten dataset is one of the most proven database used to implement, verify, and benchmark the efficacy of a majority of neural network models [29]. It is a handwritten dataset with 60,000 examples and a test set of 10,000 samples. The handwritten digits 0-9 are represented by 28x28 gray-scaled digit matrices. The database test and training set files are stored as unsigned bytes (8-bits).

In this research project, the goal is to illustrate how neural network (NN) architectures are implemented on the proven MNIST dataset by creating a two-layered Artificial Neural Network to classify the numerical values of the MNIST dataset. Each portion of the NN is built from scratch. The research project consists of deploying a primary NN cost function, cross-entropy, gradient weights, added bias, a prediction model, and one-hot encoding technique to classify each handwritten image. The results are displayed using cost and accuracy curves of the maximum accuracy achieved at a low epoch iteration experiment along with an extended epoch experiment for comparison purposes. The misclassification of the NN is also illustrated in the results section of this chapter.

4.1. Implementation

Nine samples from each numerical category were randomly selected to visualize the dataset's contents in Figure 4.1. This tiny sample illustrates the variation of 90 unique

written digits.

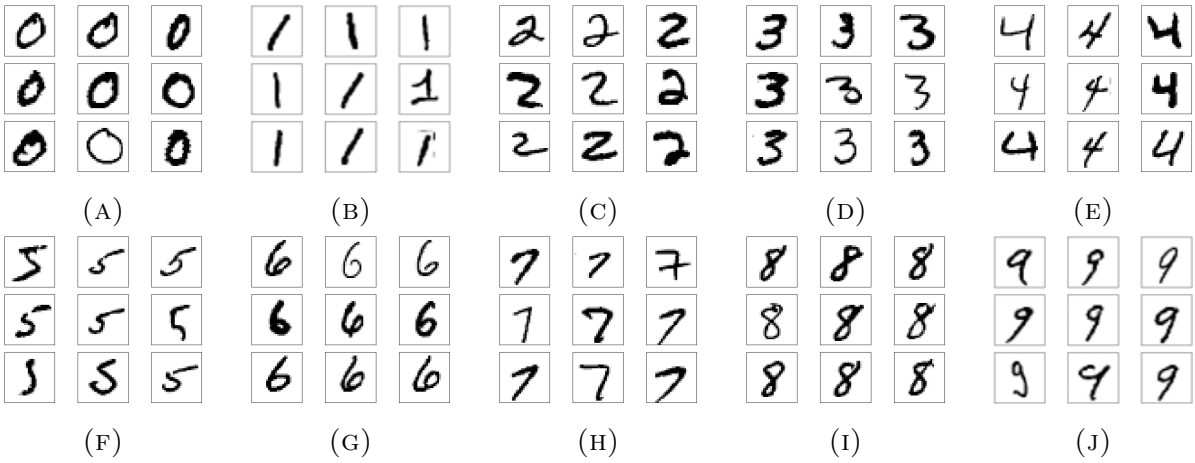


FIGURE 4.1. Randomly Selected Samples of Handwritten Numbers 0-9

The visual representation of the implemented architecture for the NN can be seen in Figure 4.4. The 28x28 matrix corresponds to the bytes of the image, flattened into a column vector array with a weight bias added. The dimensionality of each numerical image array is 785 rows by one column. The structure of the hidden layer one and two has the same number of parameters to ensure the dimensionality is maintained. Ten outputs were assigned to the output layer, given that there are ten unique handwritten digits. The error associate is calculated and then propagated backward through the model layers.

An epoch is when a NN completes an iteration over the entire dataset. Due to the limitation of today's computers, the dataset cannot be inserted into a NN's inputs all at once, so the dataset has to be divided into batches. For example, if there are 256,000 data entries and a batch size of 256, there would be 100 iterations through the data to complete one epoch. Larger batch sizes provide more computation speed, but smaller batch sizes can provide a better clarity. Overfitting is solved by generalizing data the network has not encountered. Overfitting will cause error decreases in the training set but will increase error in the test set. Early stoppage training or last save a checkpoint is implemented when performance on the validation set decreases.

The backpropagation algorithm looks for the minimum of the error function in weight space using gradient descent. The gradient is a partial derivative with respect to its inputs

for how much the output changes when the input is changed slightly; the slope of that change if the input is increased with the first function of addition, what happens to the cost function. It is the balance of increasing the inner parameters and observing what happens to function output. The combination of weights that minimizes the error function is considered a solution to the learning problem. Since this method requires the computation of the gradient of the error function at each iteration step, we must guarantee the error function's continuity and differentiability. We used the *sigmoid* activation function, explained in Chapter 2.4, rather than the step function used in perceptrons, because the composite and cost function produced by interconnected perceptrons are discontinuous. The NN cost function implemented in this research is illustrated by Equation (11) where:

- m is number of training samples, $i = 1 \cdots m$
- K is the number of nodes in the output layer, $i = 1 \cdots m$
- L is the number of layers in the network, $i = 1 \cdots m$ as we exclude the input layer
- s_l is number of nodes in a given layer l
- In the penalty term, we sum each row and column of the weight matrix. That is, from $r = 1 \cdots s_l$ and $c = 1 \cdots s_{l+1}$, r and c represent the rows and columns of the weight matrix for each layer.

$$J(\Theta) = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} log(h_\theta(x_k^{(i)})) + (1 - y_k^{(i)}) log(1 - h_\theta(x_k^{(i)})) \right] + \underbrace{\frac{\lambda}{2m} \left[\sum_{l=1}^{L-1} \sum_{r=1}^{s_l} \sum_{c=1}^{s_{l+1}} (\Theta_{r,c}^{l})^2 \right]}_{Regulator} \tag{11}$$

$H(p, q)$ is the cross-entropy function, where p is the target distribution probability, and q is approximating the target probability. Cross-entropy uses the probabilities of the events from p and q as seen in Equation (12). Since the architecture is fully connected, containing smaller nodes within each hidden layer, the exploding neuron problem is not encountered for overfeeding, so the regulator term can be omitted.

(12) $$H(p,q) = -\sum p(x)\log(q(x)), \forall x$$

The algorithm first completes a forward pass by computing the derivatives of the error function for the output layer activities k using Equation (13). Next, the backward pass calculates the error function derivatives with respect to the upper layer neurons' inputs. The weights are updated at each preceding layer until the backpropagation is completed. Algorithm 2 illustrates how the forward pass and backward pass are completed in the backpropagation process.

Algorithm 2 Backpropagation learning algorithm

1: **for** d in data **do**
2: **Forward Pass**

(13) $$g(w^T x)$$

3: **Backwards Pass**

(14) $$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

4: **for** layer in layers **do**

(15) $$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{h,k}\delta_k$$

5: **end for**
6: Updates the weights. $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

7: **end for**

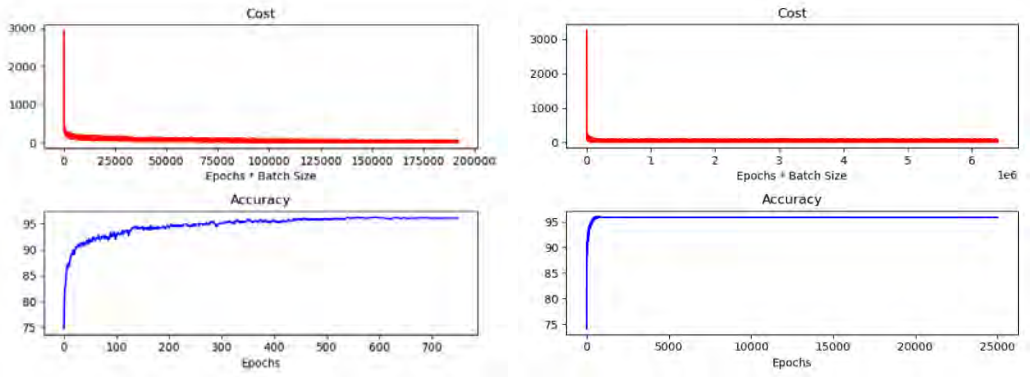
The one-hot encoding technique is applied in the algorithm to provide an order between the categories where the integer variable is substituted for a new binary variable. Table 4.1 is the corresponding lookup table for how the 0-9 samples are quantified. As the unique handwriting images are processed, the network decides which bin each respective image belongs.

TABLE 4.1. One-hot Encoding Technique for MNIST

0	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	**1**

4.2. Results

Figure 4.2 illustrates the effectiveness of the NN model design. The results in Figure 4.2a are for 750 epoch iterations which achieved an accuracy of 96.1% on the testing set. To ensure there was no error with the model, a new experiment was implemented with an epoch count of 25,000, which also achieved a 96.1% accuracy. As mentioned in Chapter 2.4, the neural network's cost function in both experiments quickly approached zero, which infers that the goal of minimization was achieved using gradient descent through the backpropagation algorithm design. Figure 4.3 displays 40 random samples of various outputs that were incorrectly classified due to the NN model's limitations.



(A) 750 epochs - 13 minutes 48 seconds training time

(B) 25,000 epochs - 9 hours 47 minutes 13 seconds training time

FIGURE 4.2. Training Performance : Batch Size - 256 - Cost vs Accuracy

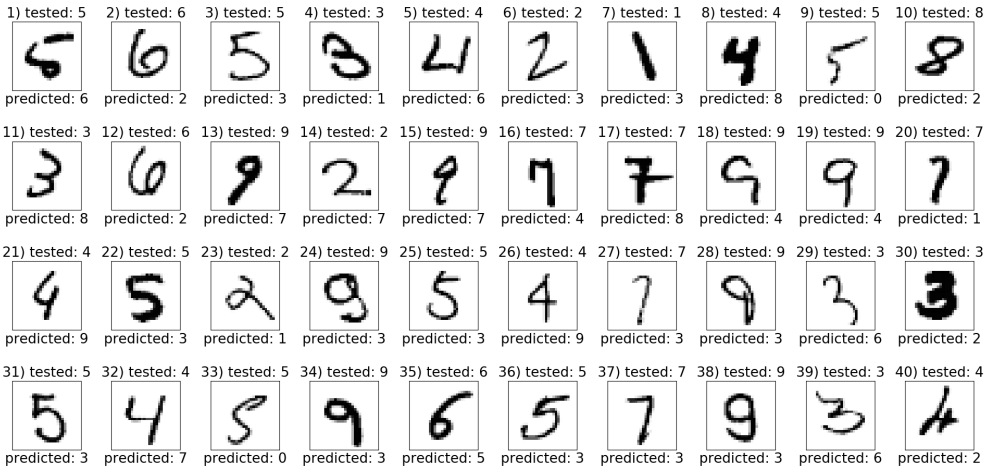


FIGURE 4.3. 40 Randomly Selected Prediction Errors

4.3. Conclusion

In this research project, a fully connected two-layered NN was created from scratch using the MNIST Handwritten dataset. The algorithm was processed on the CPU. The NN training and testing output resulted in an accuracy of 96.1% with a 750 epoch experiment. An additional experiment was completed with 25,000 epochs, and the output results for classification demonstrated an accuracy of 96.1% as well. The cost and accuracy model representations for both experiments are plotted in Figure 4.2 plots, illustrating the characteristics that closely resemble the ground truth of cost and accuracy for a NN. The misclassification of the NN shown in Figure 4.3 indicates where the algorithm could not correctly read and organize the handwritten numbers.

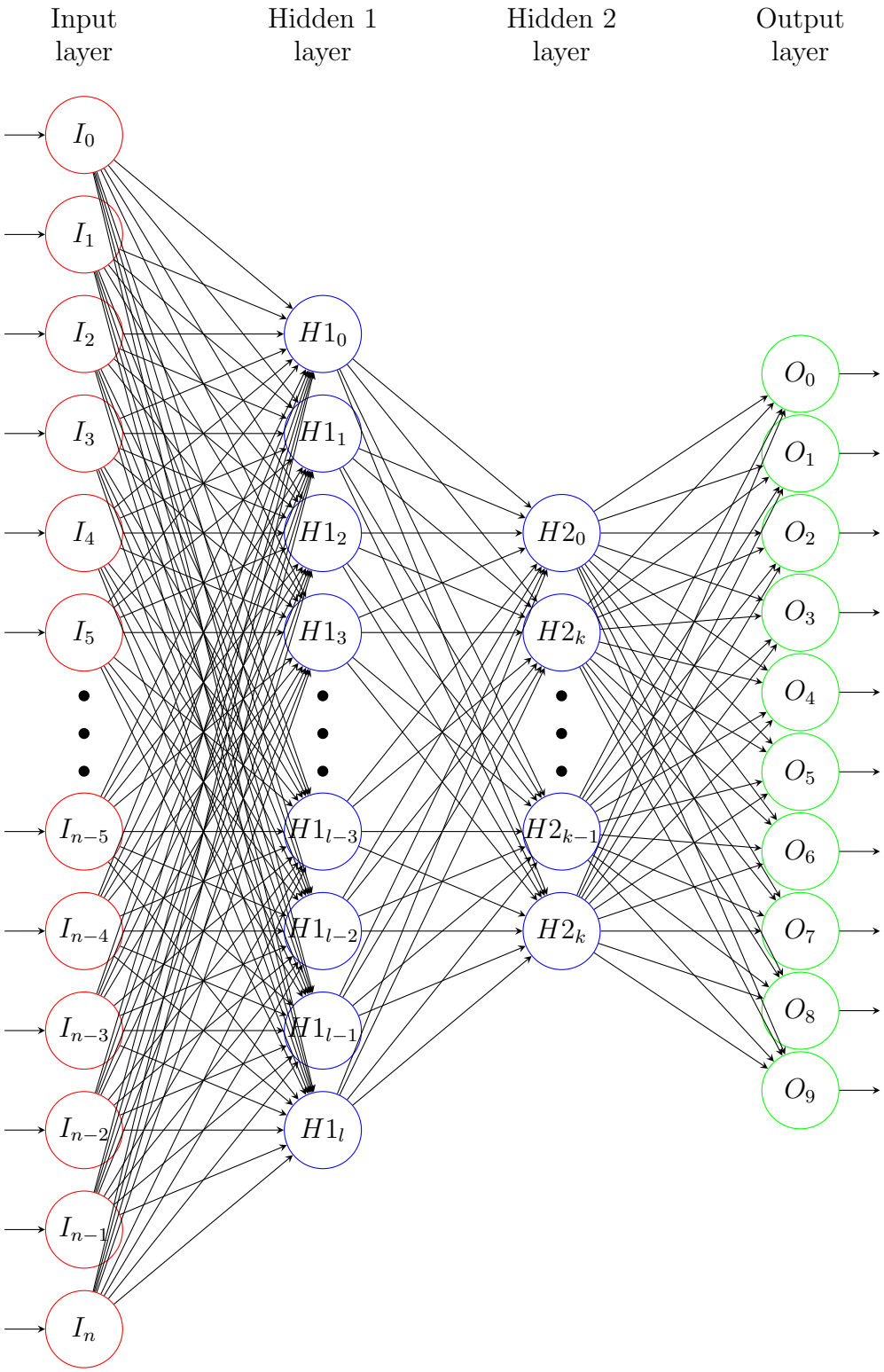


FIGURE 4.4. Neural Network Architecture: MNIST

CHAPTER 5

CONVOLUTIONAL NEURAL NETWORK FOR CLASSIFICATION OF HEARTBEATS[†]

In recent years, many fields have improved by the orders of magnitude expansion in the available data for analysis. Medicine has reaped the benefits of these advances. As with many other applications, the medical data increase has led to a refocusing of effort to consider all information at hand as valuable in solving modern medicine's demanding problems. Machine learning techniques have been applied in the medical sector and can help diagnose irregularities when data is provided for the specific area on which the system has been trained. Based on the research of the CDC [30, 31], cardiovascular diseases (CVDs) annually cause the most death globally. Analyzing the electrocardiogram (EGC) medical apparatus is the most common medical field approach to determine if a person has a cardiovascular-related disease. An EGC utilizes digital signal processing to display heartbeats on a voltage versus time graph. Health professionals can predict if people have cardiovascular diseases and diagnose based on their EGC tests to find optimal treatments for heart irregularities found in EGCs. In reality, it takes a tremendous amount of time for an expert to analyze and make the optimal solution because of the intervals between different signals created after a minuscule amount of time. To expedite the diagnosis using ECGs and reduce labor, an autonomous analyzing process using AI is highly in demand.

Moreover, with the growth of using e-health devices such as the Apple Watch, more and more data can be collected and analyzed by experts to determine the health status of people [32, 33]. Studies such as [34, 35] showed that artificial intelligence had been applied in this field by utilizing the data collected by wearable devices and provides good results in the classification of different diseases. Among all the medical datasets available, the MIT-BIH

[†]This entire chapter is reproduced from Lorenzo E. Jaques, Arthur C Depoian II, Dong Xie, Colleen P. Bailey, Parthasarathy Guturu,"A machine learning approach to medical data identification through principal component analysis," Proc. SPIE 11730, Big Data III: Learning, Analytics, and Applications, 1173003 (12 April 2021); https://doi.org/10.1117/12.2586038. Published by the Society of Photo-Optical Instrumentation Engineers (SPIE) under Creative Commons CC-BY license.

heartbeat database is considered as a benchmark when testing a medical machine learning approach to classify CVDs. Sharma et al. [36] implements a deep residual CNN orthogonal wavelet filters for differentiated erratic heartbeats. Li et al. [37] proposed a random forest in conjunction with wavelet packet entropy (WPE) grouping for the 360 Hz resolution MIT-BIH heartbeat dataset. For the 125 Hz dataset, Kachuee et al. [38] only achieved a 93.4% accuracy for stratification using a deep residual CNN. The algorithm presented in this research utilizes a multilayered, fully connected convolutional neural network to categorize electrocardiogram (ECG) data, with and without using dropout layers, achieving outstanding results.

5.1. MIT-Heartbeat Dataset

The MIT-BIH database utilized in this experiment contains 48 half-hour samples of two-lead ECG waveforms obtained while the patient was walking. The outputs were digitized at 125 Hz sampling rate and 10 mV resolution for over 109,446 recorded samples [39]. From this data, 5 different categories of waveforms were identified and labeled; normal (N), fusion (F), supraventricular ectopic (S), ventricular ectopic beats (V), and unclassifiable (Q) as seen in Table 5.1. To better visualize each signal, a random sample from each category can be seen in Figure 5.1.

TABLE 5.1. Summary of mappings between beat annotations and AAMI EC57 [1] categories.

Category	**Annotations**
N	-Normal -Left/Right bundle branch block -Atrial escape -Nodal escape
S	-Atrial prematrure -Aberrant Atrial premature -Nodal Premature -Supra-ventricular premature
V	-Premature ventricular contraction -Ventricular escape
F	-Fusion of Ventricular and normal
Q	-Paced -Fusion of paced and normal -Unclassifiable

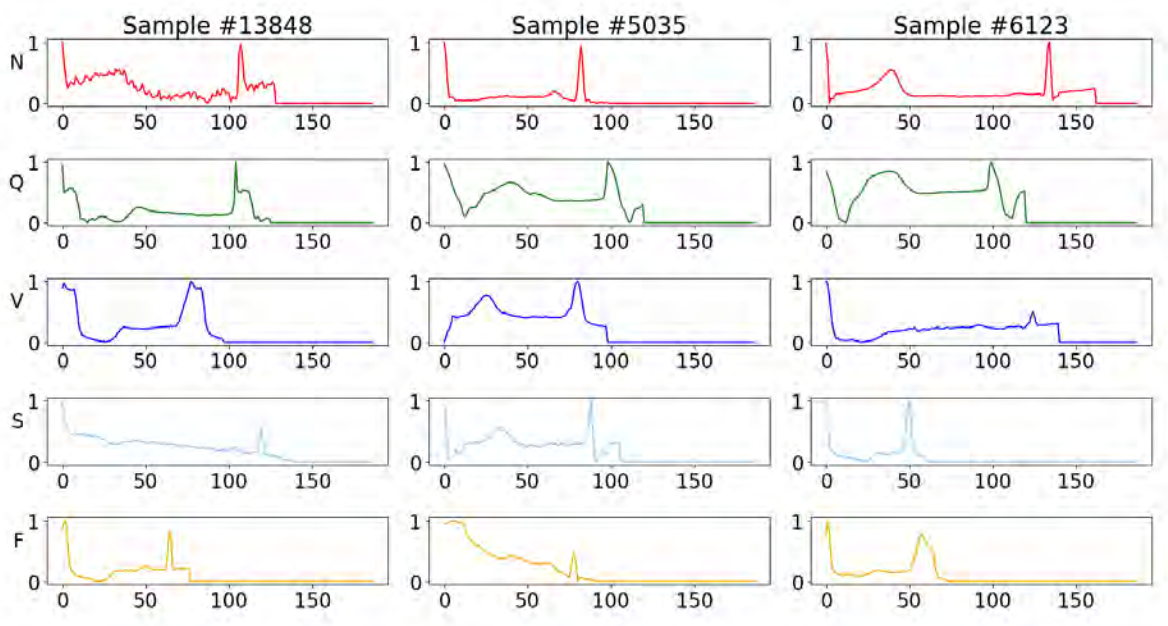


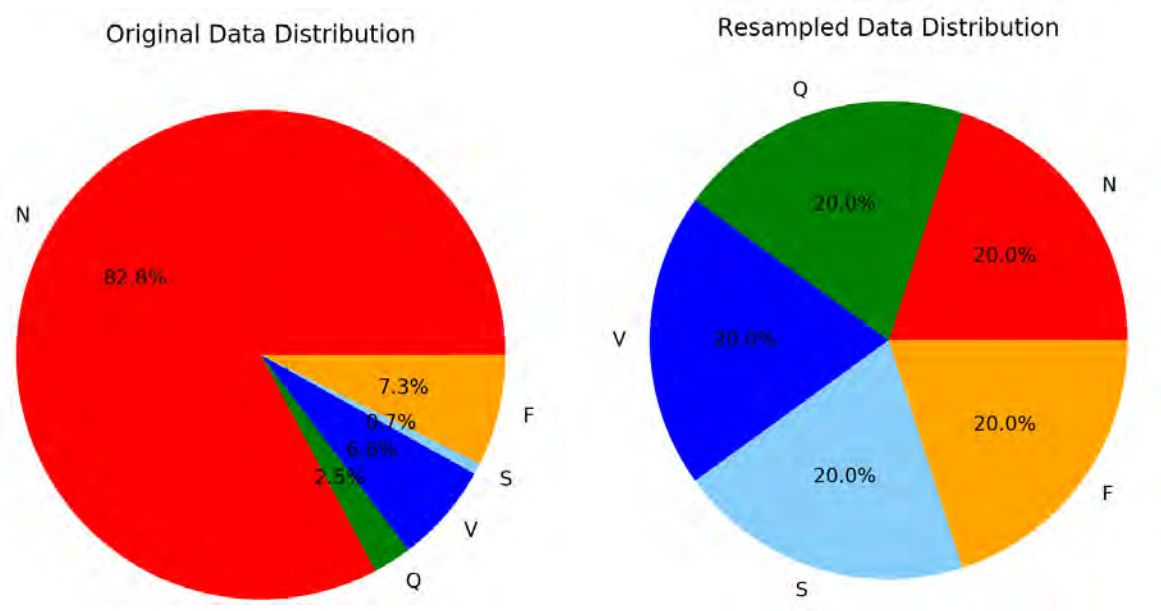
FIGURE 5.1. Random Samples

Original data balancing is required before the neural network input layer. The majority of data pertains to normal beats, as shown in Figure 5.2a. The reshaping of the dataset is crucial to the neural network structure. If the input were the original class distribution of the data, most of the training time would be spent on the 'Normal Heartbeat' category and not enough on the other four categories. To address this issue, the dominant majority class dataset, Normal Heartbeat, was subsampled by extracting random samples from the dominant class to a count of 20,000. The remaining four categories were expanded by adding a set weight value for the model parameter for the neural network's inner connection. The subsample factor is multiplied by the original example weight to equal the final example weight of each remaining weight in their respective category. Each equates to 20,000 per class with the redistribution illustrated in Figure 5.2b. The applied balancing ensures the outputs are interpreted equally to mitigate skewed results. The pseudo-code for the balancing is expressed in Algorithm 3. The neural network was fed in these image samples and classified each identified beat into five distinct categories.

The split data is spread to the inputs of the entire original data neural network model, one dropout layer neural network model, and two dropout layers network model at every

Algorithm 3 Preprocessing the Data

```
procedure LOAD MIT-BIH DATABASE
    Convert to numpy array
    Shape array into 1 Dimension
    Designate desired weights of each category
        Resample each category class to 20,000 samples
end procedure
```



(A) Original Distribution

(B) Re-sampled Distribution

FIGURE 5.2. Distribution of Database

epoch interval to maintain the comparison's integrity as shown in Figure 5.3. Each of the proposed methods is evaluated with the exact test implementation to equally validate the three separate networks' efficacy. Each distinct network is exposed to the same data for training and testing each experimental epoch iteration. The completion logs of the parallel experiments were stored for comparative analysis.

5.1.1. Model

The array stack is reshaped and then flattened for all 188 data inputs. The array stack manipulation is then fed in on its first axis of the data stacked from the row-major form. The next layer convolves the array stacks to compare between the various layers to funnel down into the next layer where the max-pooling occurs. The sample-based discretization

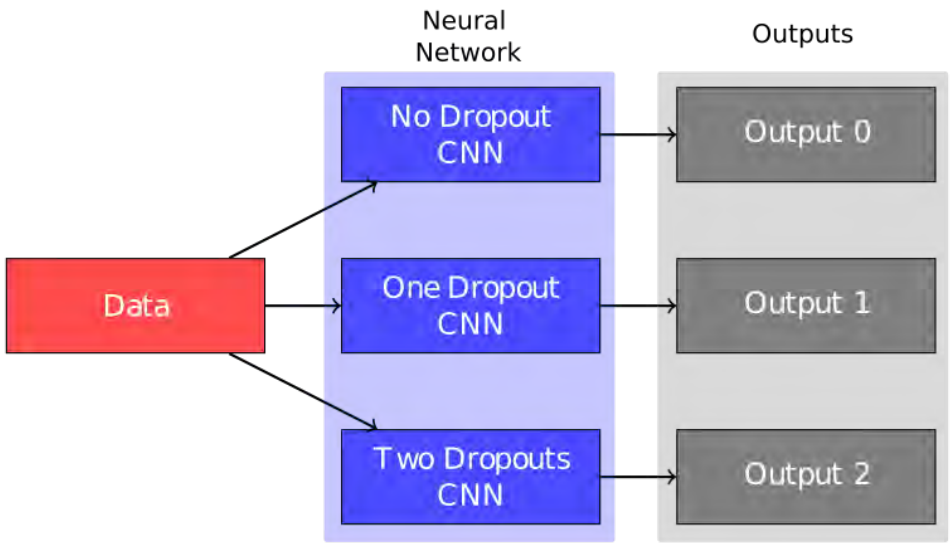


FIGURE 5.3. Global Architecture

process orients the numerical evaluation and implementation to compare the array stacks leftover from the convolution process. The dense layer following has each input node connected to each output node to keep continuity between the data arrays. A loss function is used for regularization, and the algorithm seeks to reduce this as much as possible. The implemented CNN contains 3x1-D convolutional layers, each with a max-pool step immediately after. Then this data is fed to 3 dense layers, decreasing in size, to the final output of five dimensions, one for each classification label. An early stop callback was implemented to have an arbitrarily large number of training epochs to end the training once the performance model started to degrade for the classification's output. The early stopping execution was crucial to avoid overfitting and mitigate overtraining the neural network before picking up statistical noise. If there is an excess of statistical noise, the model becomes useless for the sake of categorization and classification. A visual representation of the neural network can be seen in Figure 5.4.

5.2. MIT-Heartbeat Dataset - Results

To implement the CNN, the Tensorflow CUDA framework was used, having all the heavy-duty complex computation executed on the GPU. Table 5.2 shows the preliminary

Algorithm 4 1D Convolution Neural Network Design

procedure LOAD RESAMPLED DATA
    Convolution 64 x 6
        Max-Pool 3 x 2
    Convolution 64 x 3
        Max-Pool 2 x 2
    Convolution 64 x 3
        Max-Pool 2 x 2
    Dense ends applied
    Validation loss check state
end procedure

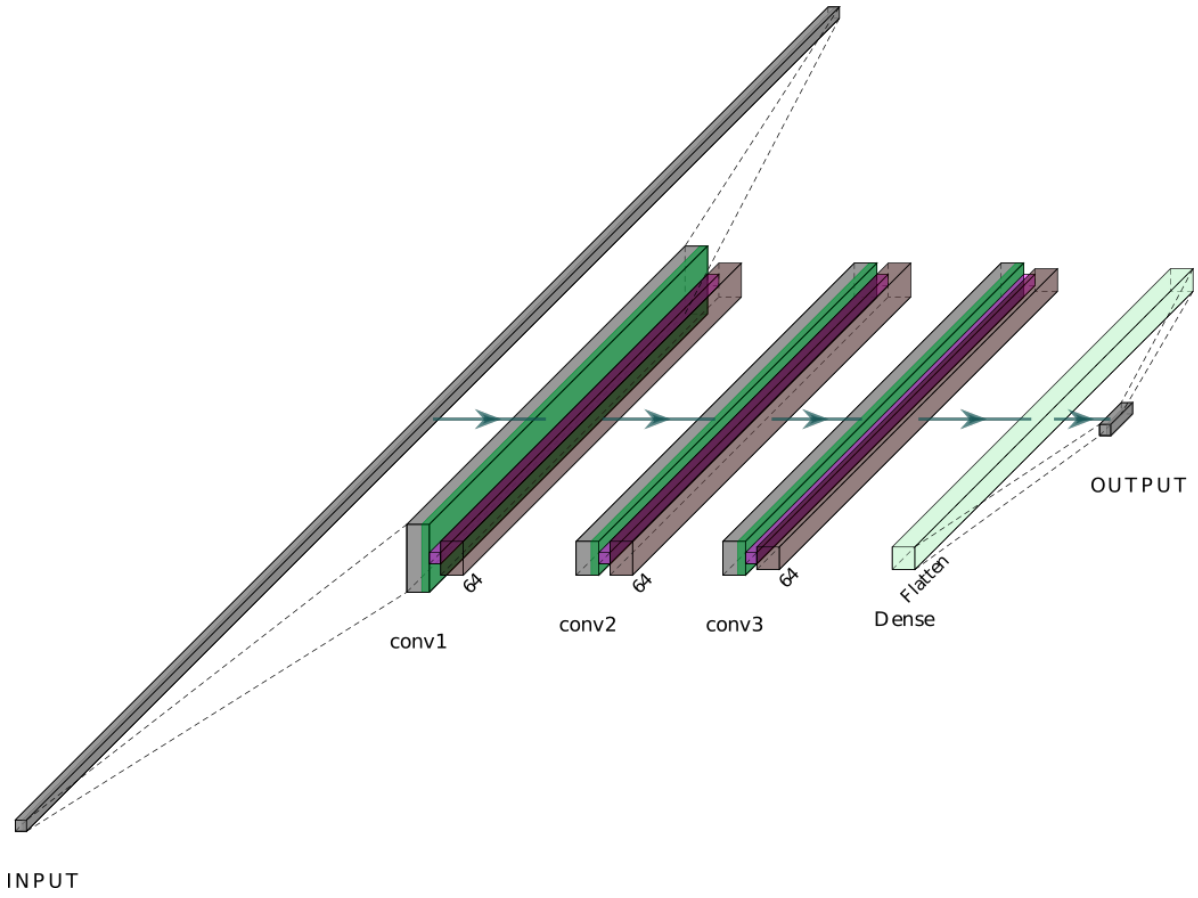
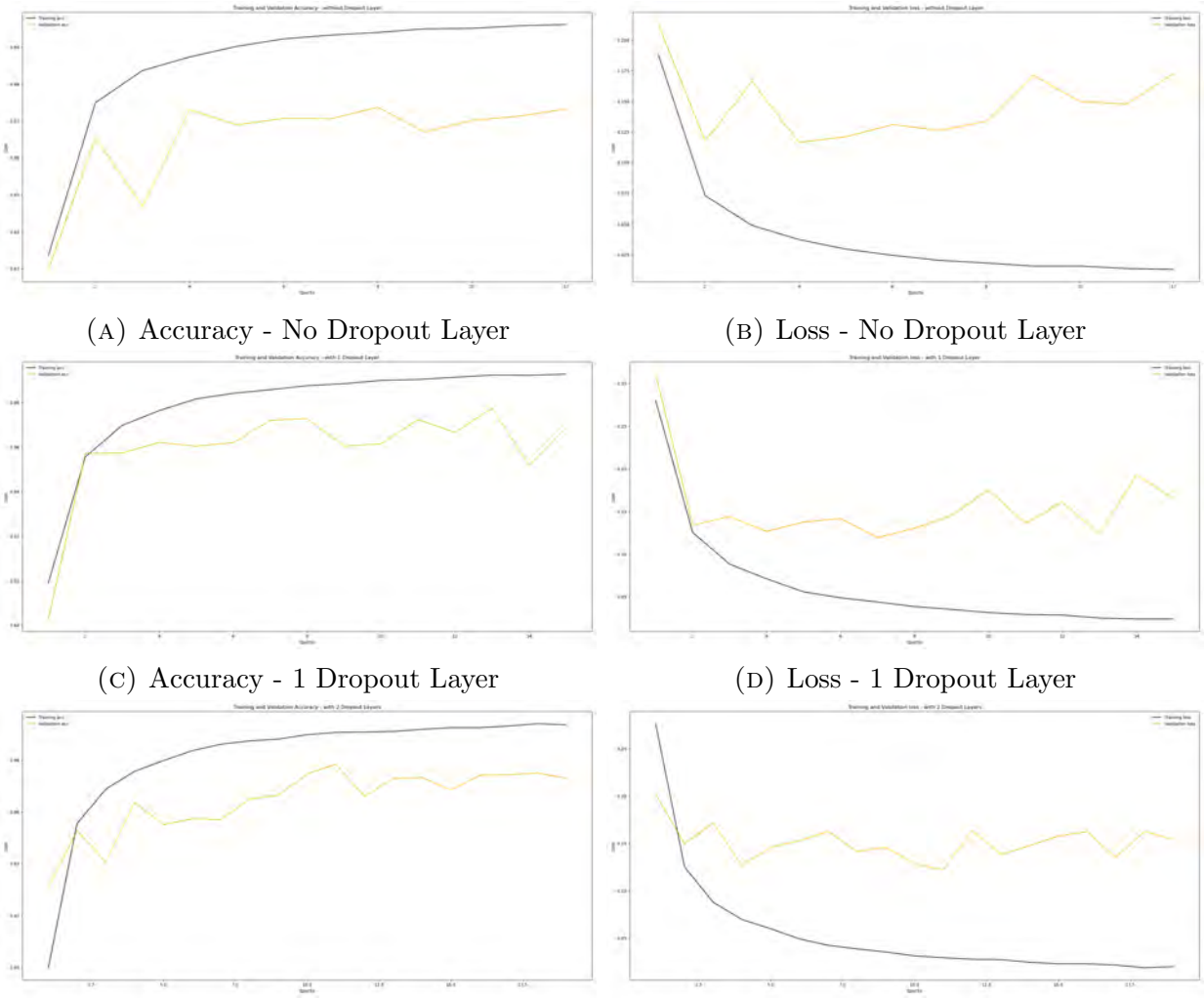


FIGURE 5.4. Medical Convolutional Neural Network Architecture

results of each class accuracy in the train class of **one** experiment for No Dropout Layer Network, One Dropout Layer Network, and Two Dropout Layers network models. Figure 5.5 are the visual representation of how each respected neural network performed for the same **one** experiment. The smaller the delta distance between the Training Accuracy line (black) and the Validation Accuracy line (orange), the better our created CNN's overall



(A) Accuracy - No Dropout Layer

(B) Loss - No Dropout Layer

(C) Accuracy - 1 Dropout Layer

(D) Loss - 1 Dropout Layer

(E) Accuracy - 2 Dropout Layers

(F) Loss - 2 Dropout Layers

FIGURE 5.5. Training Performance : Batch Size - 512
Training Accuracy/Loss (Black) Validation Accuracy/Loss (Orange) - 1 Experiment Sample

performance and reliability. The same concept applies to the Training Loss and Validation Loss for having confidence that the network performs as intended when applied to the testing data set.

Initially, 25 experiments were completed for the three models with epoch iteration checkpoints of 10, 20, 30, 40, & 50. The results of the experiments between the No Dropout CNN, One Dropout CNN Layer, and Two Dropout Layers CNN can be seen in Table 5.3. The training and testing showed promising results, with the average of all 25 training experiments were recorded. The results concluded that the No Dropout Layer CNN performed the best

TABLE 5.2. Training & Testing Comparison Between Networks - 1 experiment

0 Dropout Layers

Training

Label	Normal	Unknown	Ventricular Ectopic	Supraventricular Ectopic	Fusion
Total Number	20566	19458	19859	20104	20013
Correct Number	19746	19347	19684	19849	19934
Predict Accuracy	96.01%	99.43%	99.12%	98.73%	99.61%

Testing

Label	Normal	Unknown	Ventricular Ectopic Beats	Supraventricular Ectopic Beats	Fusion
Total Number	17869	639	1476	245	1654
Correct Number	17739	458	1377	145	1584
Predict Accuracy	99.27%	71.67%	93.29%	57.09%	95.77%

1 Dropout Layer

Training

Label	Normal	Unknown	Ventricular Ectopic Beats	Supraventricular Ectopic Beats	Fusion
Total Number	20165	20106	19927	19810	19992
Correct Number	19738	19917	19860	19688	19860
Predict Accuracy	97.88%	99.06%	99.66%	99.38%	99.82%

Testing

Label	Normal	Unknown	Ventricular Ectopic Beats	Supraventricular Ectopic Beats	Fusion
Total Number	17787	762	1495	204	1643
Correct Number	17684	476	1389	140	1591
Predict Accuracy	99.42%	62.47%	92.91%	68.29%	96.84%

2 Dropout Layers

Training

Label	Normal	Unknown	Ventricular Ectopic Beats	Supraventricular Ectopic Beats	Fusion
Total Number	20060	19926	19840	20174	20000
Correct Number	19902	19873	19816	20000	19988
Predict Accuracy	99.21%	99.73%	99.88%	99.14%	99.94%

Testing

Label	Normal	Unknown	Ventricular Ectopic Beats	Supraventricular Ectopic Beats	Fusion
Total Number	17955	651	1444	201	1638
Correct Number	17834	475	1372	142	1598
Predict Accuracy	99.33%	72.96%	95.01%	69.61%	97.56%

TABLE 5.3. Average - Training & Testing Comparison Between Networks - 25 experiments

Training						
Dropout Layers	Epochs	10	20	30	40	50
0	Average Train Raw	99.13%	99.58%	99.57%	99.83%	99.97%
1	Average Train Raw	99.22%	99.43%	99.68%	99.59%	99.80%
2	Average Train Raw	99.08%	99.58%	99.66%	99.73%	99.76%

Testing						
Dropout Layers	Epochs	10	20	30	40	50
0	Average Test Raw	97.03%	97.31%	97.52%	97.86%	97.98%
1	Average Test Raw	97.02%	97.37%	97.66%	97.71%	97.82%
2	Average Test Raw	96.78%	97.45%	97.54%	97.67%	97.66%

at 97.98%, but further exploration of the high average accuracy score was explored.

The results obtained were from completing 125 experiments of the trained model epoch iteration checkpoints set to 25, 50, 75, and 100. The peak performance for over 125 distinct neural network models for No Dropout CNN shows the possibilities of what the neural network structure can do at 125 Hz resolution in Table 5.4. The peak results illustrated are comparable results with other peak performance results reported with a higher resolution of 360 Hz for the MIT-BIH dataset. While the peak performance is a fantastic metric, a more realistic representation of the network proposed is the average scores for universally used benchmarks as seen in Table 5.5. The weighted accuracy, overall recall, overall precision, overall F1, and weighted F1 score are listed to explain the results further. The "overall" averages the metric regardless of the data distribution, while "weighted" metrics consider the data distribution.

Yes, accuracy is essential, but it does not consider false positives, true positives, false negatives, and true negatives. To further solidify our results, the recall and precision are reported showing the potential of misclassification. The F1 score is the proper metric for how a network's performance is due to the balance needed between precision and recall. The F1 utilizes both precision and recall in the calculation. Specifically, the weighted F1 score illustrates each category's performance while simultaneously taking precision and accuracy into account. F1 scores are stressed in results due to these experiments pertaining to the

TABLE 5.4. Peak Metrics - Training & Testing Comparison - No Dropout Layer - 125 experiments

Training				
Epochs	25	50	75	100
Weighted Accuracy	100.00%	100.00%	100.00%	100.00%
Overall Recall	100.00%	100.00%	100.00%	100.00%
Overall Precision	100.00%	100.00%	100.00%	100.00%
Overall F1	100.00%	100.00%	100.00%	100.00%
Weighted F1	100.00%	100.00%	100.00%	100.00%

Testing				
Epochs	25	50	75	100
Weighted Accuracy	98.15%	98.22%	98.26%	98.27%
Overall Recall	93.12%	93.23%	93.27%	93.18%
Overall Precision	90.64%	90.35%	90.38%	90.27%
Overall F1	90.71%	91.75%	91.41%	91.49%
Weighted F1	98.16%	98.24%	98.28%	98.29%

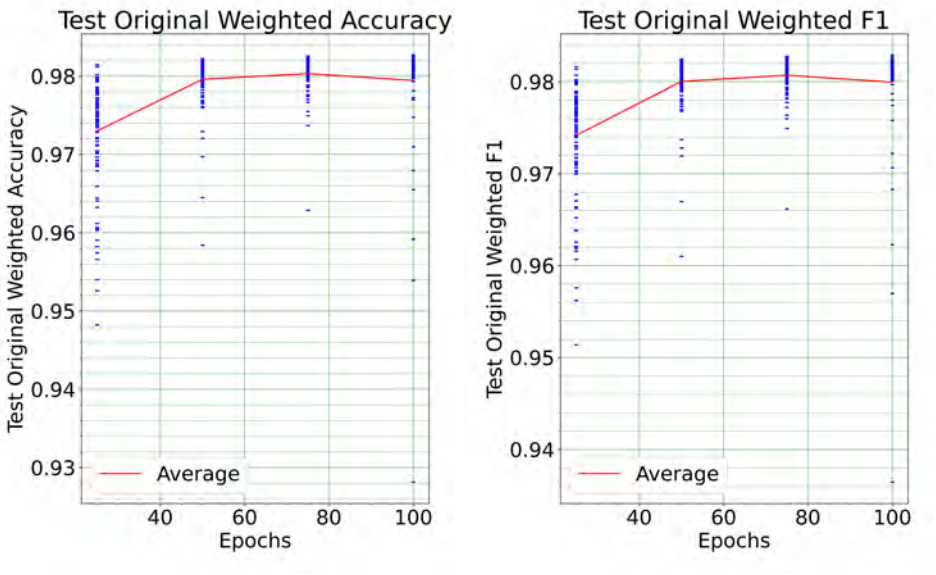
TABLE 5.5. Average Metrics - Training & Testing Comparison - No Dropout Layer - 125 experiments

Training				
Epochs	25	50	75	100
Weighted Accuracy	99.19%	99.89%	99.98%	99.90%
Overall Recall	99.19%	99.89%	99.98%	99.90%
Overall Precision	99.26%	99.89%	99.98%	99.90%
Overall F1	99.19%	99.89%	99.98%	99.90%
Weighted F1	99.19%	99.89%	99.98%	99.90%

Testing				
Epochs	25	50	75	100
Weighted Accuracy	97.30%	97.96%	98.03%	97.94%
Overall Recall	91.50%	92.14%	92.29%	92.23%
Overall Precision	85.43%	88.19%	88.55%	88.23%
Overall F1	88.01%	90.01%	90.31%	90.05%
Weighted F1	97.42%	98.00%	98.07%	98.00%

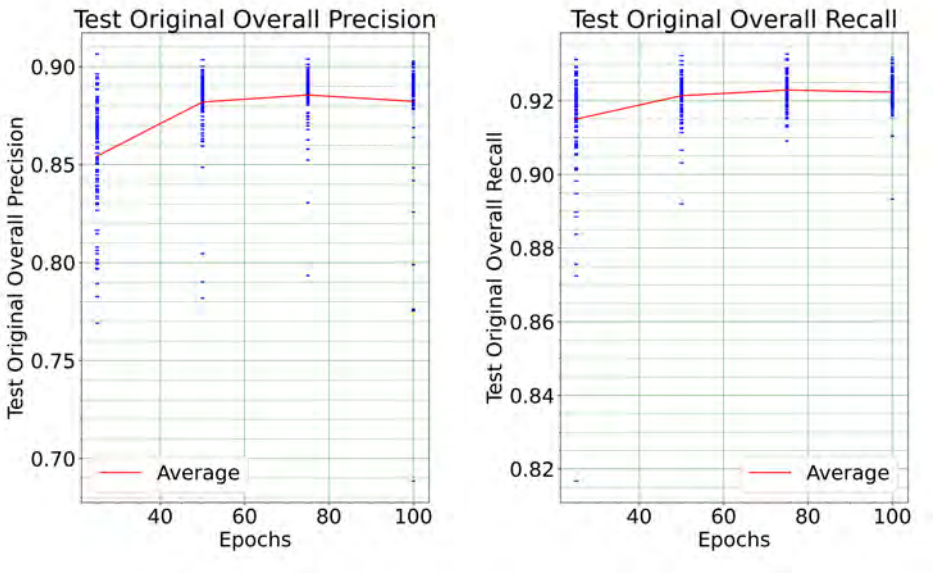
medical industry. False negatives and false positives can be deadly in some cases. As more epoch iterations were fed through the model, the model displayed in Figure 5.6a shows a positive growth over time, substantially increasing to just under 97.94% for the weighted average and an average weighted F1 score of 98%. As seen in the four metric plots in Figure 5.7, each blue dot represents one neural network model completing an incremental epoch

checkpoint with 125 blue dots on each respective epoch iteration designated interval. The red trend line is the average of the corresponding plot line displaying an upward trend. Every plot's metric curve indicates a positive upward trend across the board on the No Dropout CNN testing phase.



(A) Accuracy (B) F1 Score

FIGURE 5.6. Average Testing Metrics - Accuracy & F1 Score



(A) Precision (B) Recall

FIGURE 5.7. Average Testing Metrics - Precision & Recall

5.3. Conclusion

In this research project, three distinct fully connected CNNs, No Dropout Layer, One Dropout Layer, and Two Dropout Layers, were implemented using the Tensorflow framework with the complex processing completed on the GPU. Initial results in Table 5.3 show that the best performing network was the No Dropout Layer CNN after 25 experiments using the same train and test data for each of the three networks. Further experiments were implemented on the No Dropout CNN to a total of 125 unique fully connected CNN experiments. The peak metrics in Table 5.4 show the capabilities of the trained model. The weighted F1 scores and weighted accuracy from Table 5.5 illustrated phenomenal results with an average weighted accuracy score of 97.94% and an average weighted F1 score of 98% to validate the extensive testing of over the 125 models experiments.

CHAPTER 6

CONCLUSION

In the presented research, a color quantization compression was implemented using the USC-SIPI image dataset. The results illustrated an optimal value of cluster points paired with compression results indicating that using the kmeans algorithm implemented could reasonably be an excellent way to compress images while maintaining high quality. The new flip metric from the summer of 2020 from the NVIDIA R & D was implemented on all tested images. The ANN presented in chapter 4 performed at parity with other backpropagation algorithms with a 96.1% accuracy at a 750 experiment and was validated through a 25,000 epoch experiment. Three CNNs were implemented in parallel using the MIT-BIH heartbeat dataset at a 125 Hz sampling rate with a 10 mV resolution. Still, the CNN with no dropout layers performed the best of the three after 25 experiments. Additional research was done on the no dropout layer CNN where 125 experiments were completed achieving a weighted F1 score of 98% and a weighted accuracy score of 97.94% beating out many other models that earned a lower accuracy score with the 360 Hz MIT-BIH heartbeat dataset.

APPENDIX

RESULTS - CHAPTER 3

A.1. Results - Kmeans - Airplane

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

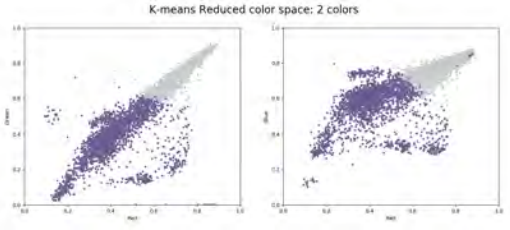
(F) 64-color clusters

(G) 128-color clusters

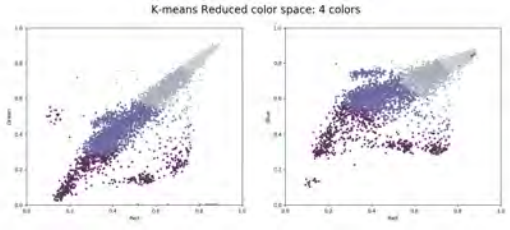
(H) 256-color clusters

(I) Original Image

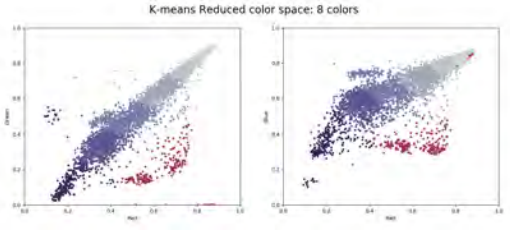
FIGURE A.1. Airplane - Various Kmeans color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



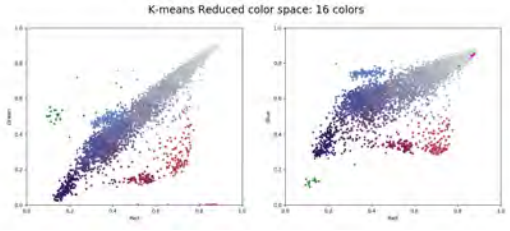
(A) 2 colors



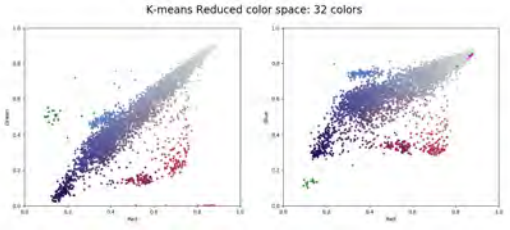
(B) 4 colors



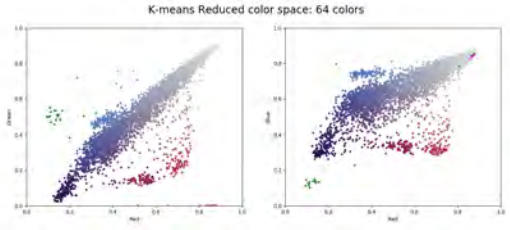
(C) 8 colors



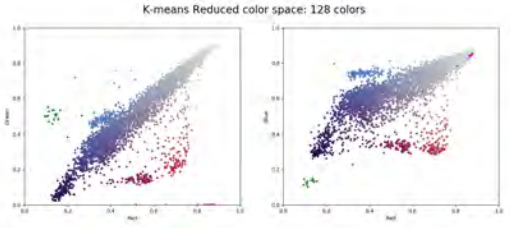
(D) 16 colors



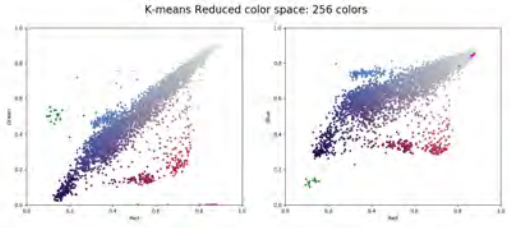
(E) 32 colors



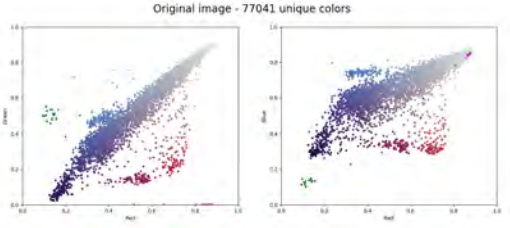
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 77041 unique colors

FIGURE A.2. Airplane - Various Kmeans color scatter plots (2, 4, 8, 16, 32, 64, 128, 256, & 77041)

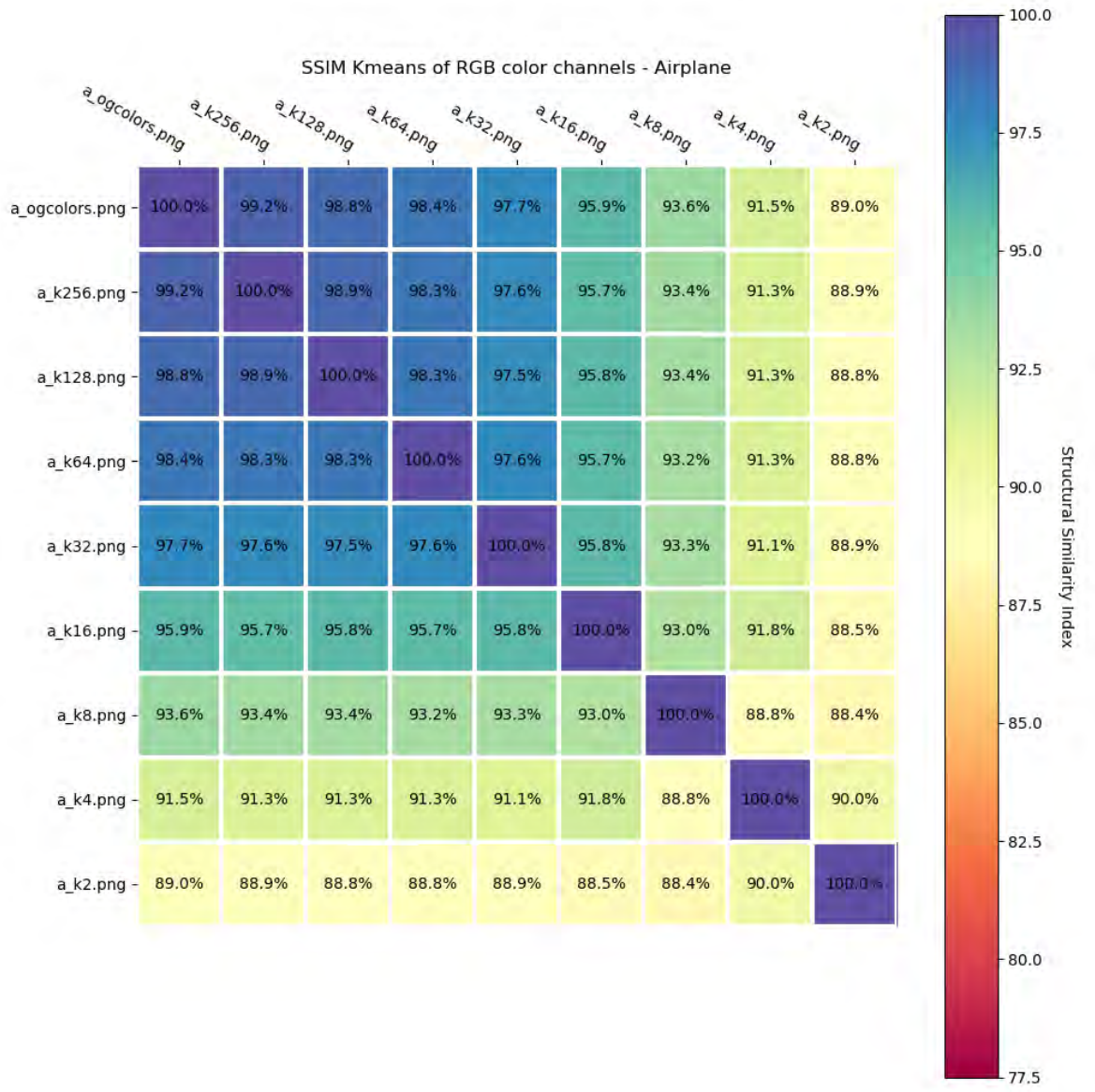


FIGURE A.3. SSIM Heatmap - Kmeans RGB color channels - Airplane

A.2. Results - Random Color Selection - Airplane

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

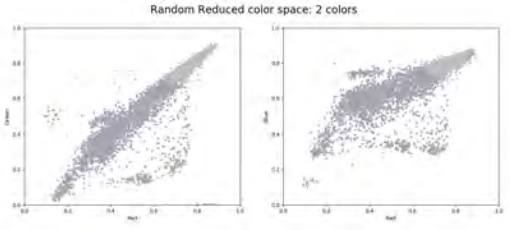
(F) 64-color clusters

(G) 128-color clusters

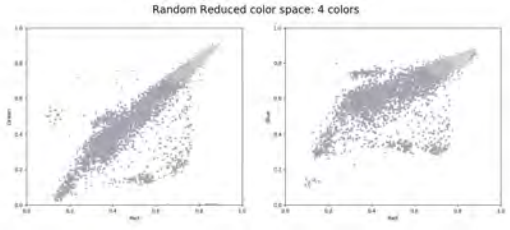
(H) 256-color clusters

(I) Original Image

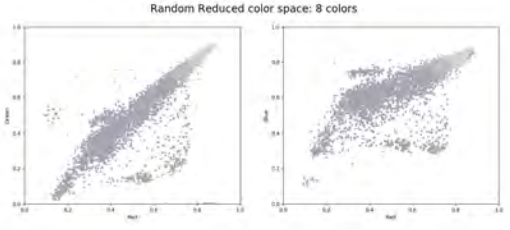
FIGURE A.4. Airplane - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



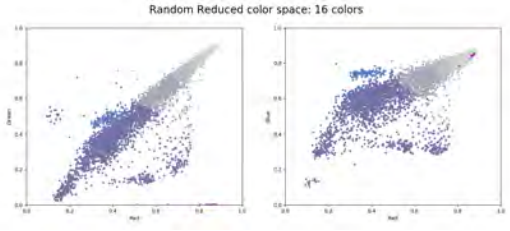
(A) 2 colors



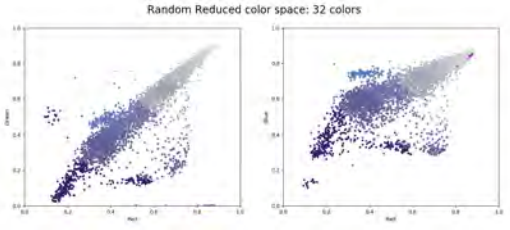
(B) 4 colors



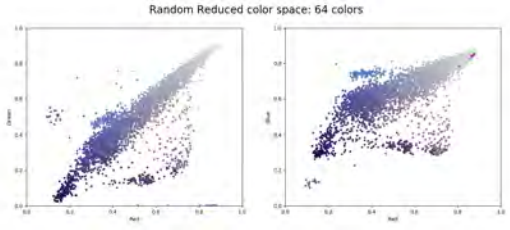
(C) 8 colors



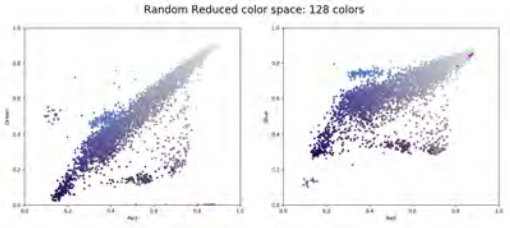
(D) 16 colors



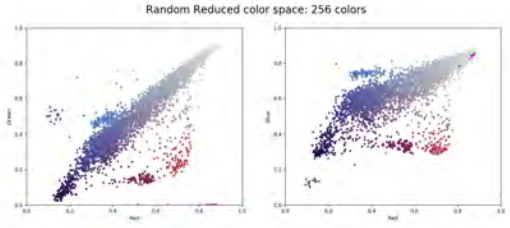
(E) 32 colors



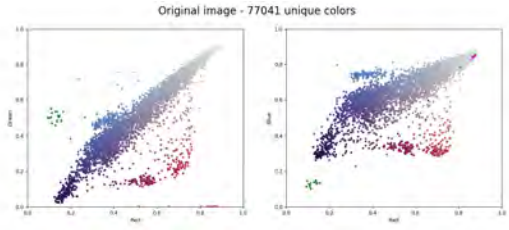
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 77041 colors

FIGURE A.5. Airplane - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & 77041)

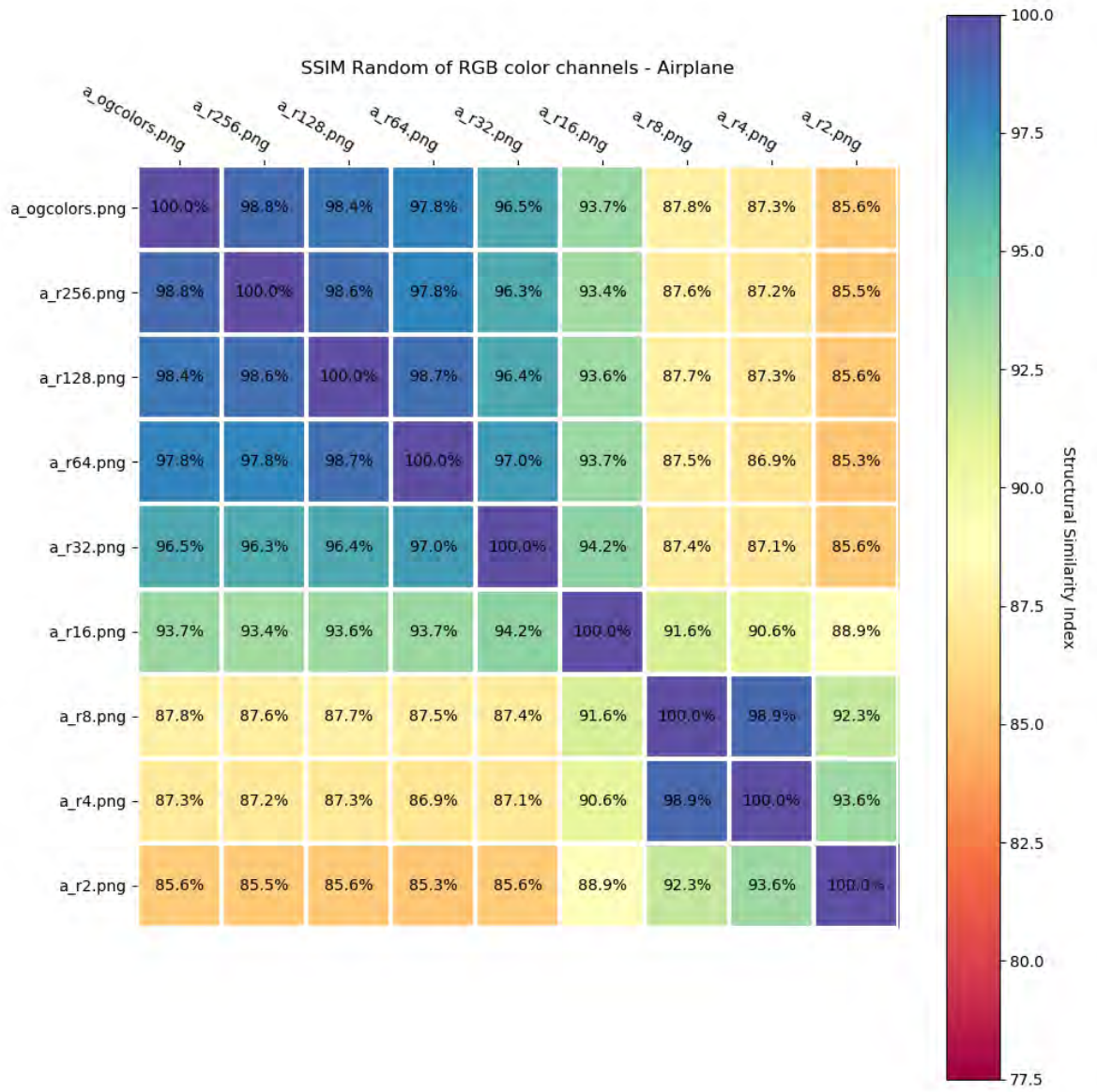
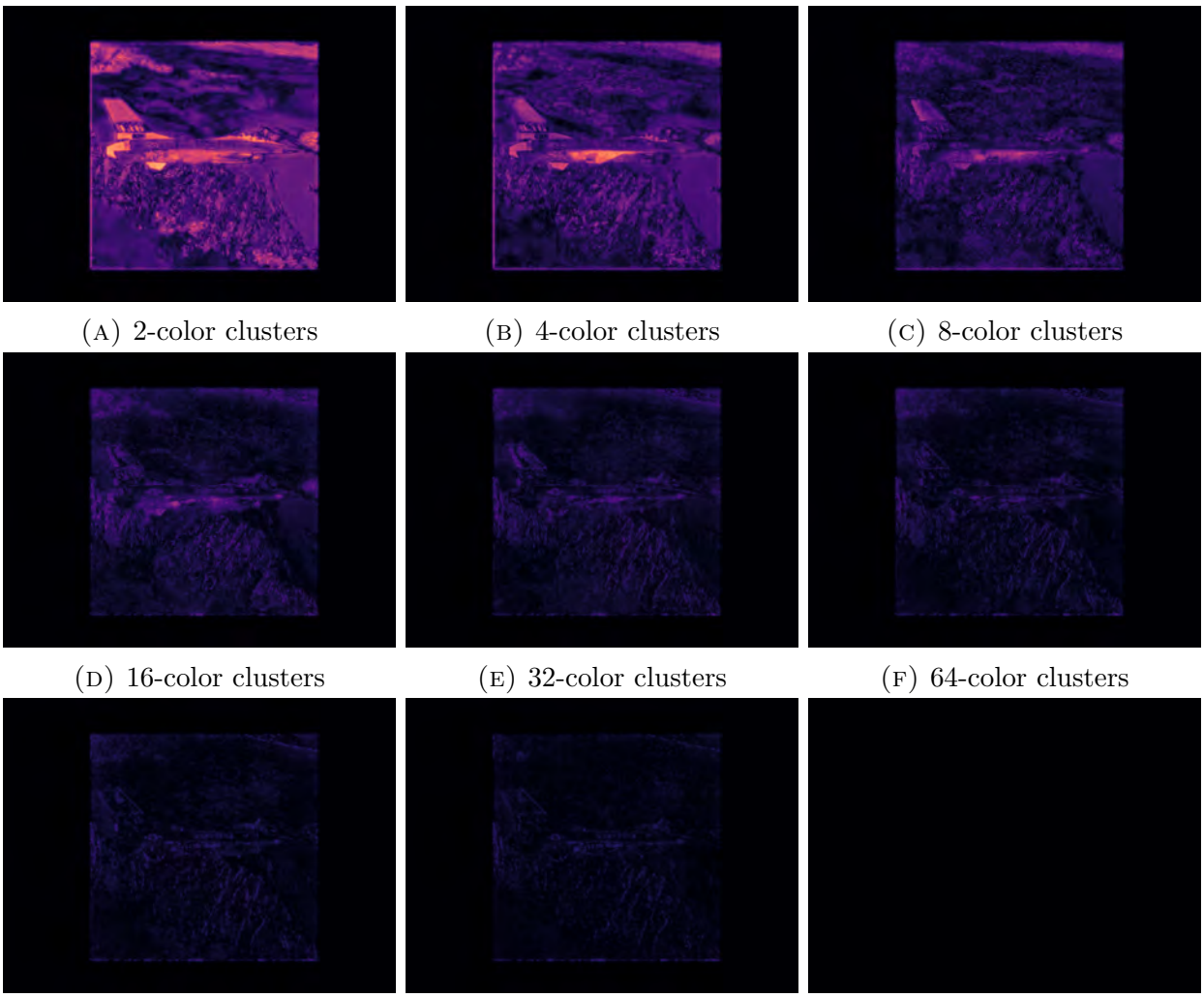


FIGURE A.6. SSIM Heatmap - Random RGB color channels - Airplane



(A) 2-color clusters (B) 4-color clusters (C) 8-color clusters

(D) 16-color clusters (E) 32-color clusters (F) 64-color clusters

(G) 128-color clusters (H) 256-color clusters (I) Original Image

FIGURE A.7. Airplane - ꟻLIP Kmeans color clusters (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

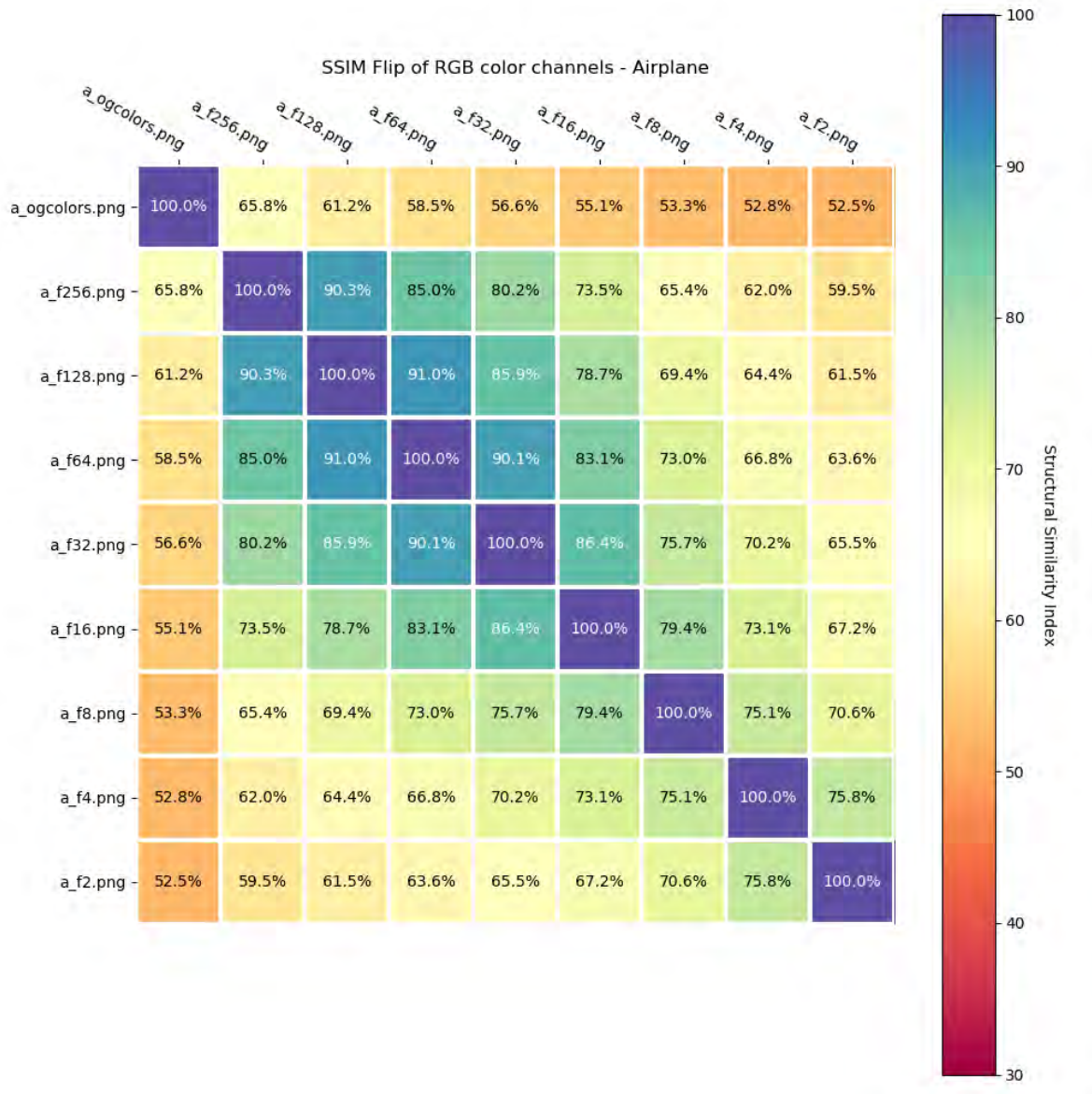


FIGURE A.8. SSIM Heatmap - ꟻLIP Kmeans RGB color channels - Airplane

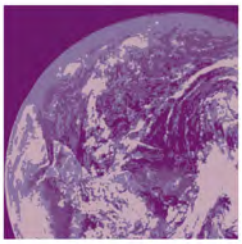
A.4. Results - Storage Space - Airplane

TABLE A.1. Airplane Compression Comparison

	Storage Size	
Cluster Size	Kmeans	Random Selection
Original	283.9 kB	283.9 kB
256	257.7 kB	259.9 kB
128	219.0 kB	239.1 kB
64	167.0 kB	202.8 kB
32	115.4 kB	153.5 kB
16	66.9 kB	94.5 kB
8	38.9 kB	56.7 kB
4	26.4 kB	36.3 kB
2	10.8 kB	11.9 kB

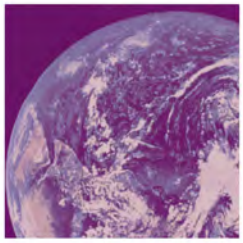
A.5. Results - Kmeans - Earth

(A) 2-color clusters

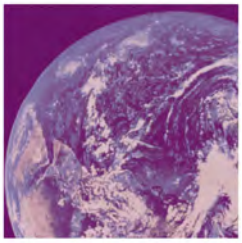


(B) 4-color clusters

(C) 8-color clusters



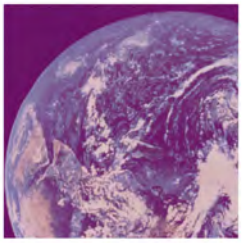
(D) 16-color clusters



(E) 32-color clusters

(F) 64-color clusters

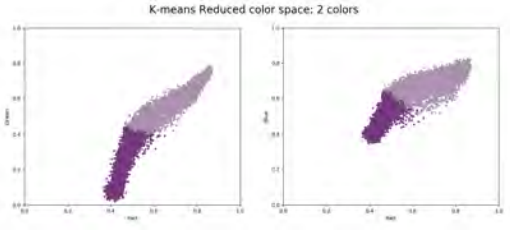
(G) 128-color clusters



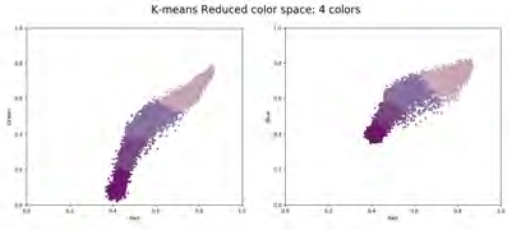
(H) 256-color clusters

(I) Original Image

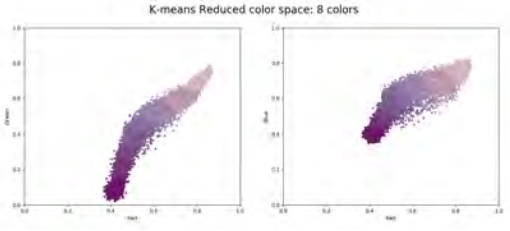
FIGURE A.9. Earth - Various Kmeans color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



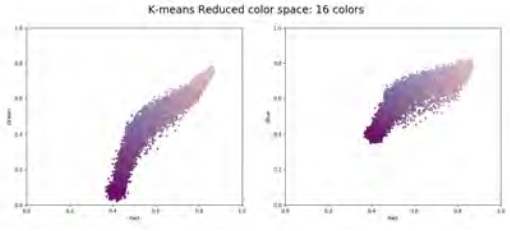
(A) 2 colors



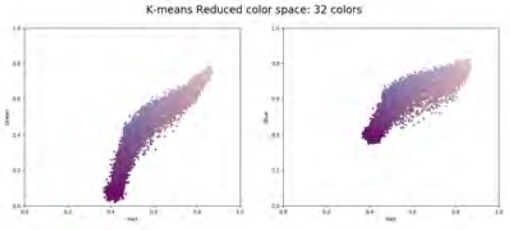
(B) 4 colors



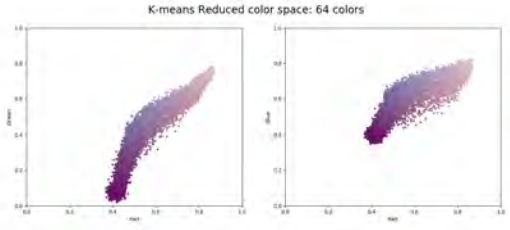
(C) 8 colors



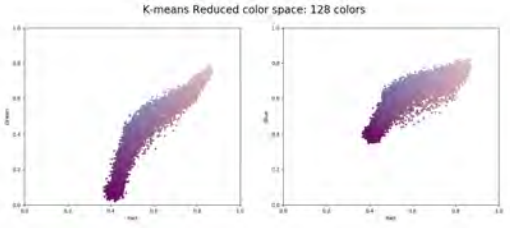
(D) 16 colors



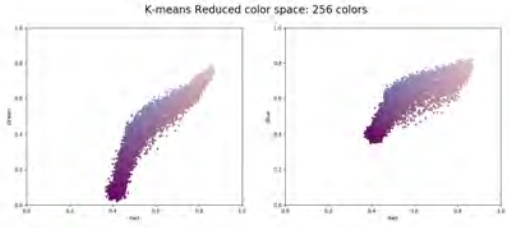
(E) 32 colors



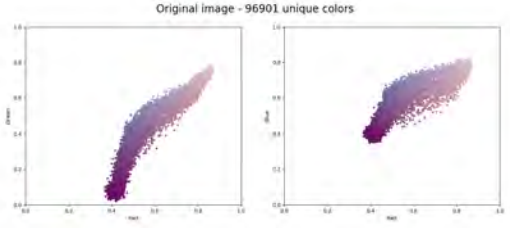
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 96901 unique colors

FIGURE A.10. Earth - Various Kmeans color scatter plots (2, 4, 8, 16, 32, 64, 128, 256, & 96901)

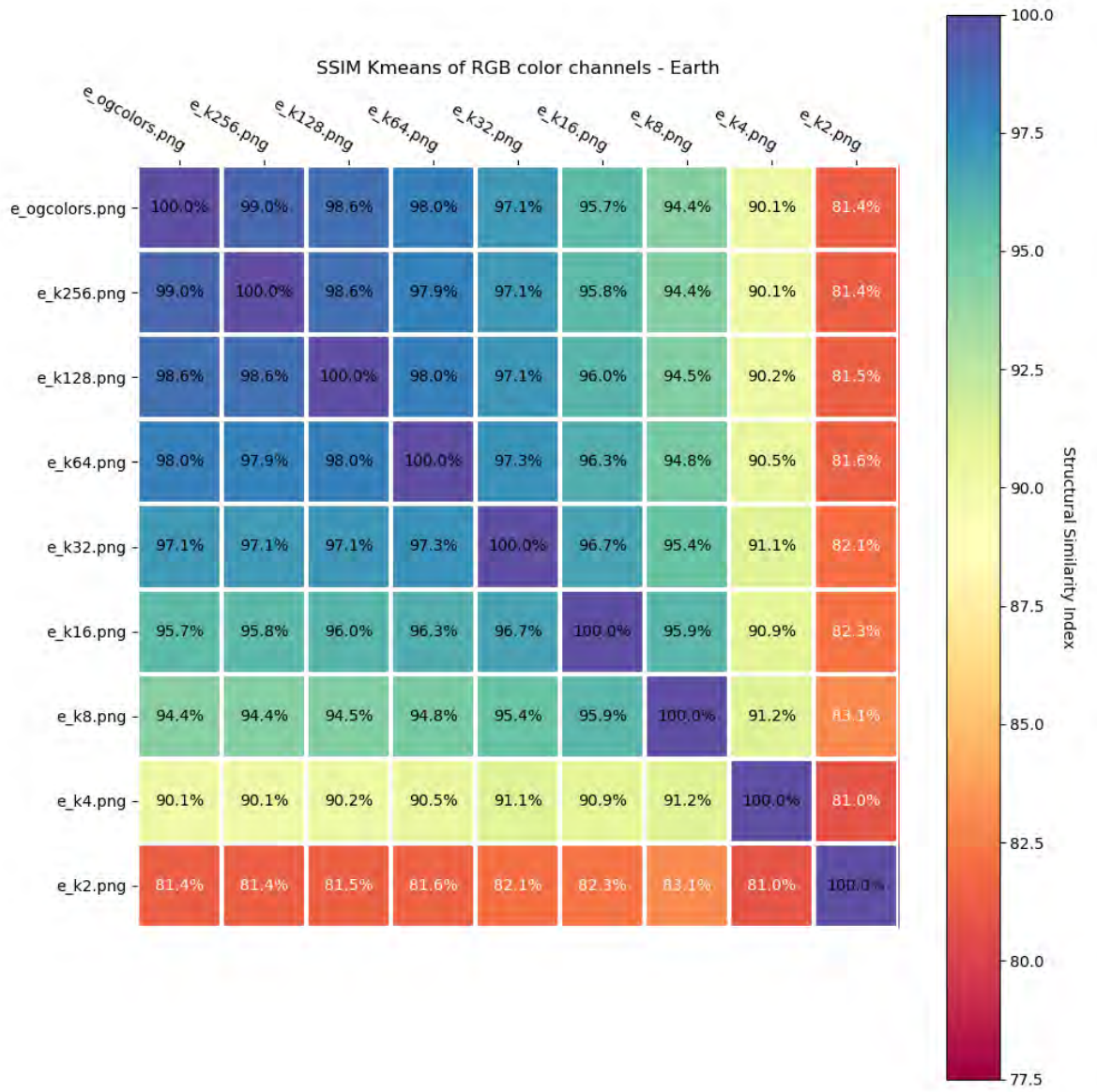
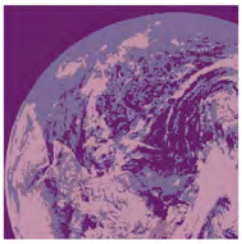


FIGURE A.11. SSIM Heatmap - Kmeans RGB color channels - Earth

(A) 2-color clusters

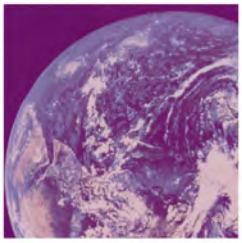


(B) 4-color clusters

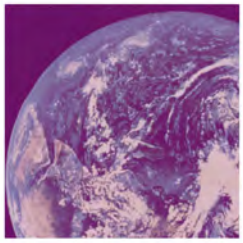
(C) 8-color clusters

(D) 16-color clusters

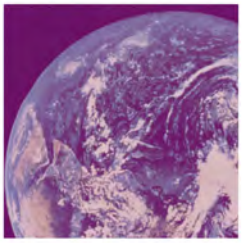
(E) 32-color clusters



(F) 64-color clusters



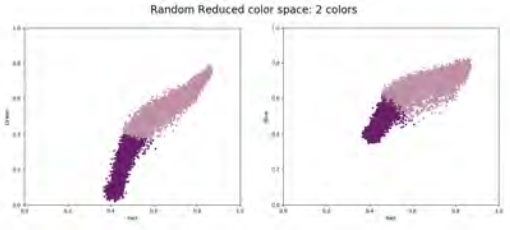
(G) 128-color clusters



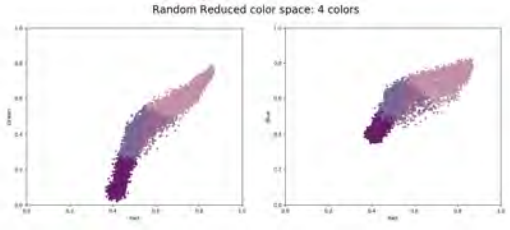
(H) 256-color clusters

(I) Original Image

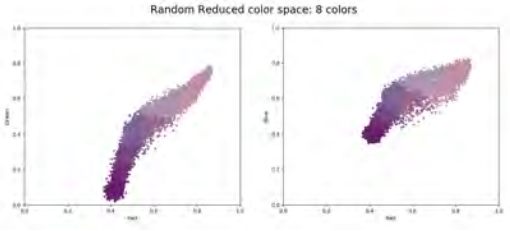
FIGURE A.12. Earth - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



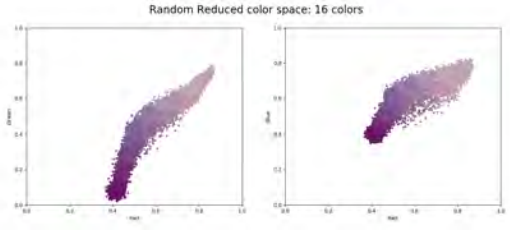
(A) 2 colors



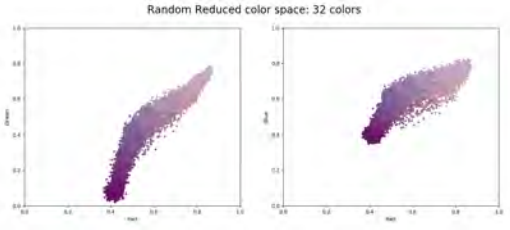
(B) 4 colors



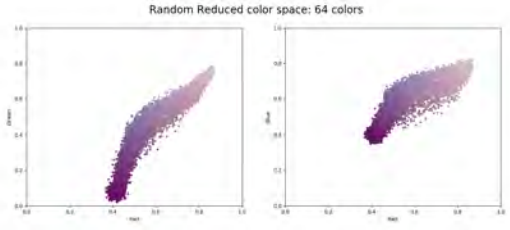
(C) 8 colors



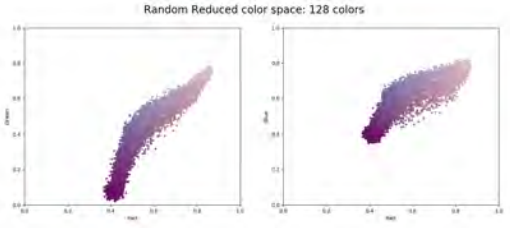
(D) 16 colors



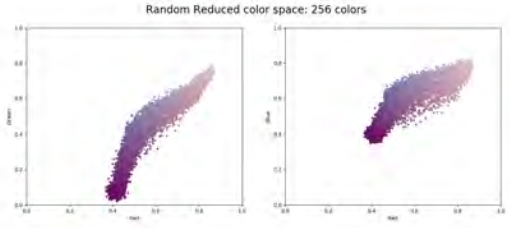
(E) 32 colors



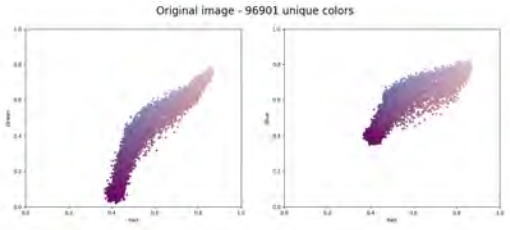
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 96901 unique colors

FIGURE A.13. Earth - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & 96901)

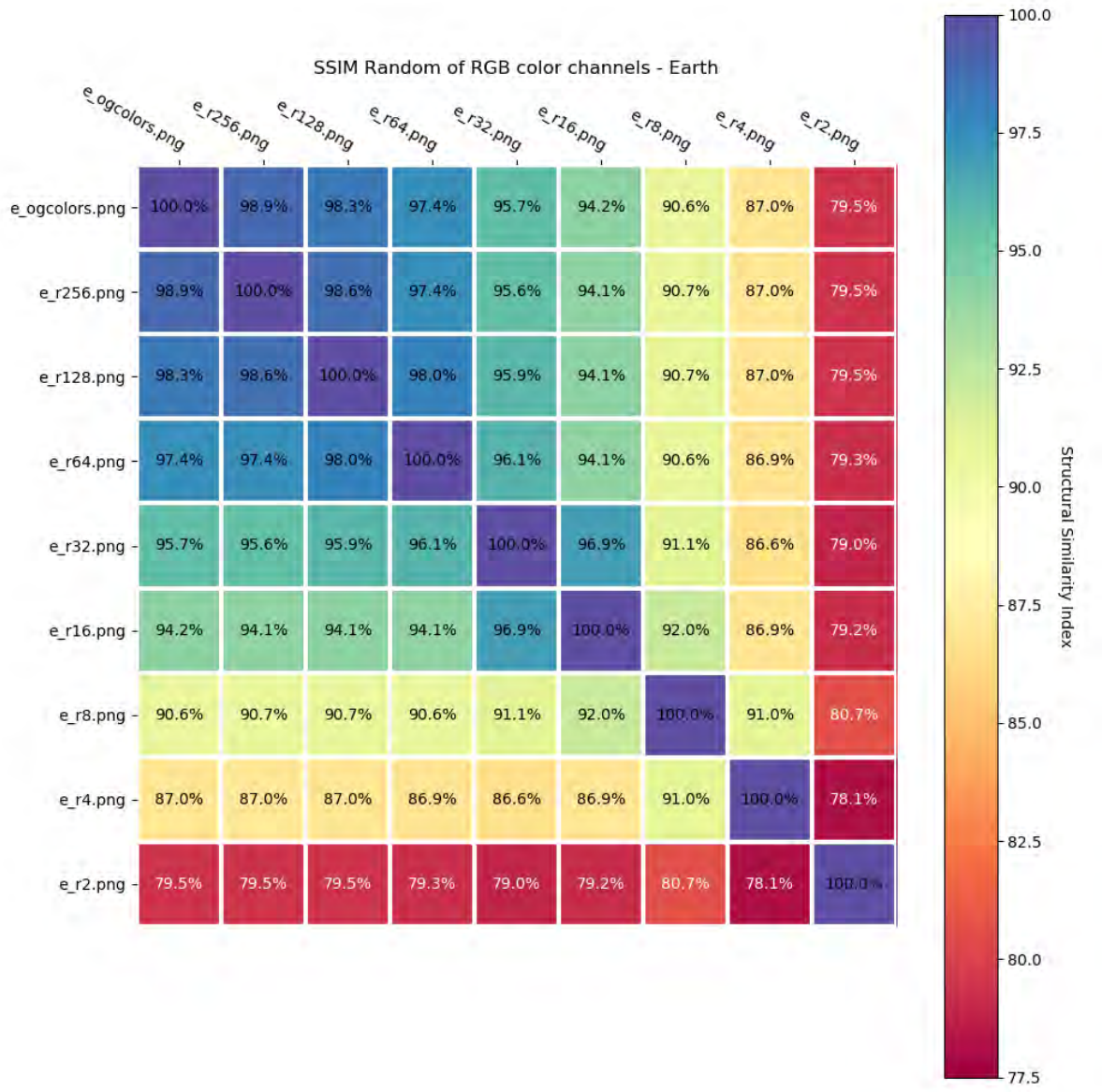
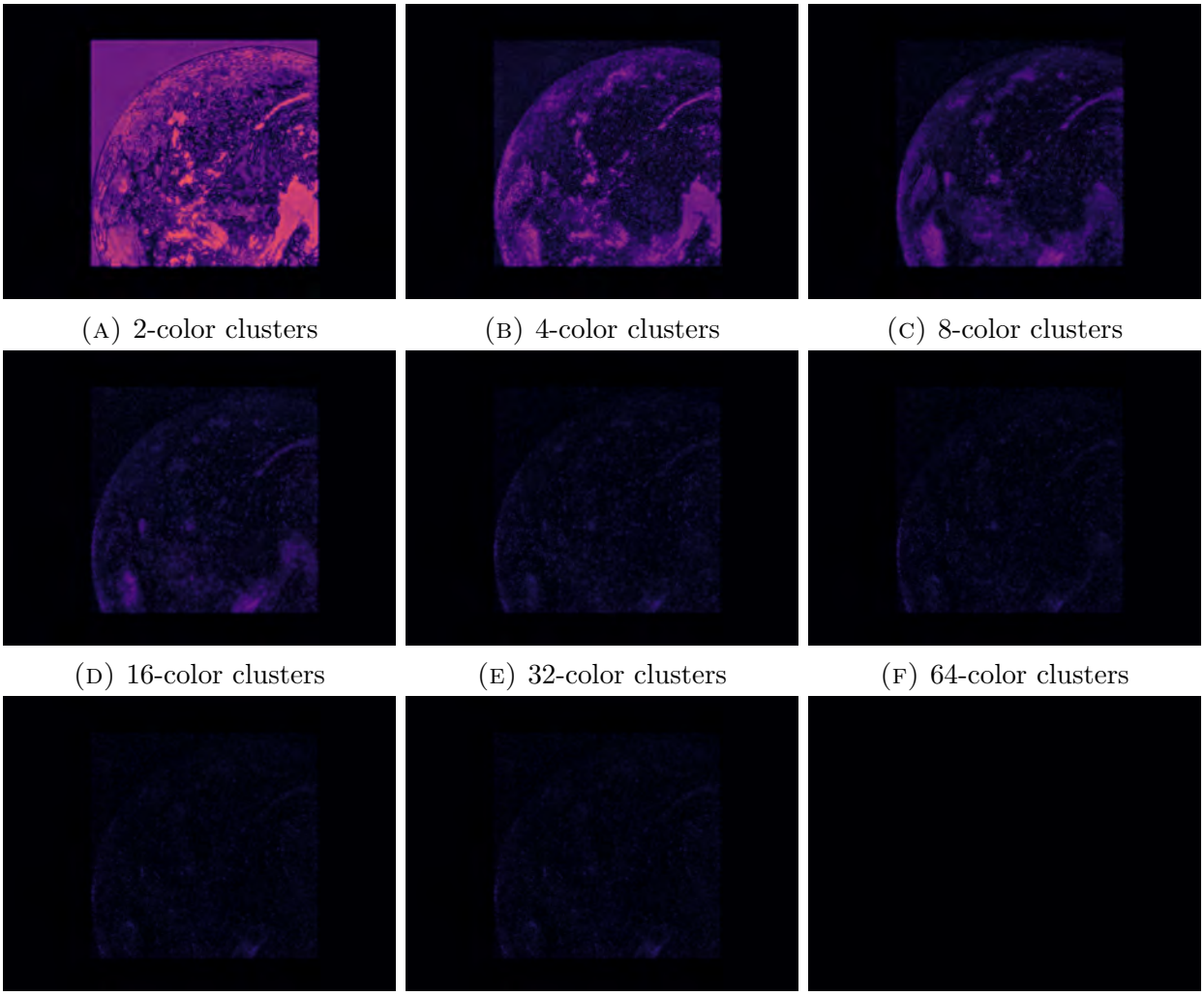


FIGURE A.14. SSIM Heatmap - Random RGB color channels - Earth

A.7. ꟻLIP Results - Earth



(A) 2-color clusters (B) 4-color clusters (C) 8-color clusters

(D) 16-color clusters (E) 32-color clusters (F) 64-color clusters

(G) 128-color clusters (H) 256-color clusters (I) Original Image

FIGURE A.15. Earth - ꟻLIP Kmeans color clusters (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

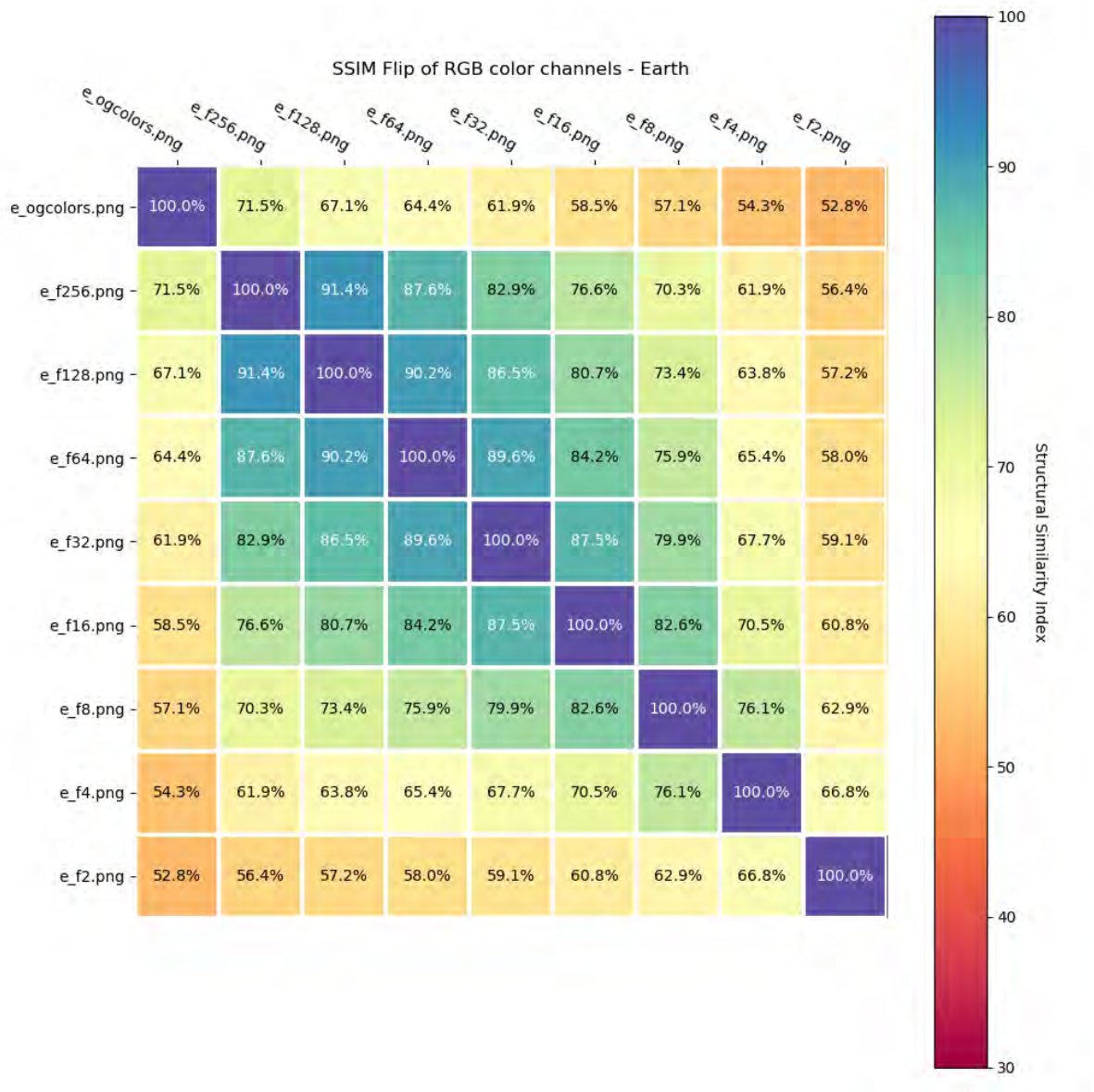


FIGURE A.16. SSIM Heatmap - ꟻLIP Kmeans RGB color channels - Earth

TABLE A.2. Earth Compression Comparison

	Storage Size	
Cluster Size	Kmeans	Random Selection
Original	324.7 kB	324.7 kB
256	317.9 kB	315.9 kB
128	302.0 kB	301.0 kB
64	264.8 kB	263.0 kB
32	197.7 kB	200.9 kB
16	117.3 kB	128.1 kB
8	71.9 kB	71.5 kB
4	36.7 kB	37.2 kB
2	19.1 kB	19.2 kB

A.9. Results - Kmeans - Sailboat

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

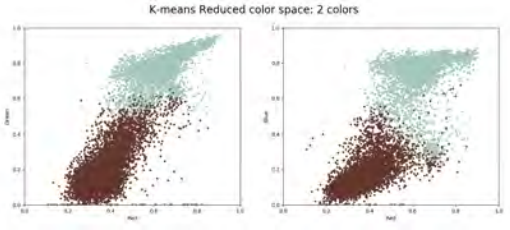
(F) 64-color clusters

(G) 128-color clusters

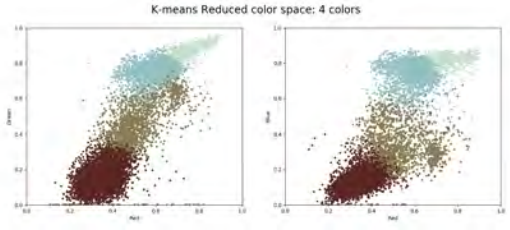
(H) 256-color clusters

(I) Original Image

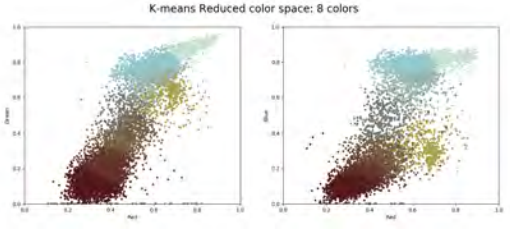
FIGURE A.17. Sailboat - Various Kmeans color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



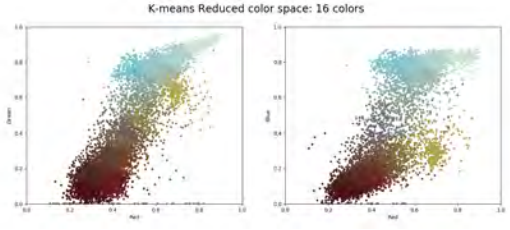
(A) 2 colors



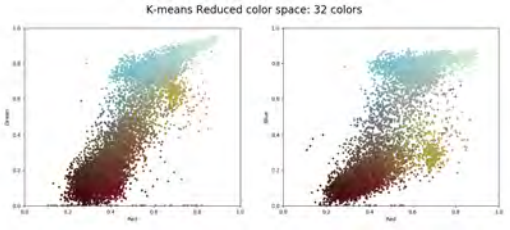
(B) 4 colors



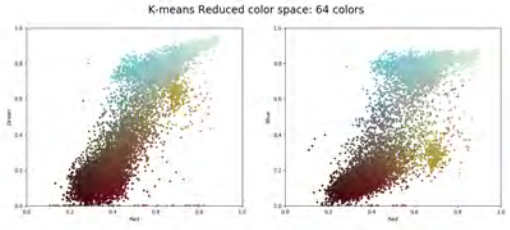
(C) 8 colors



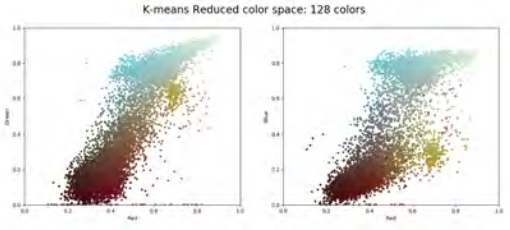
(D) 16 colors



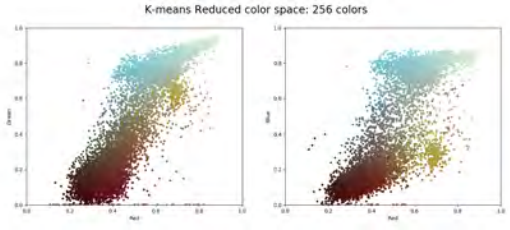
(E) 32 colors



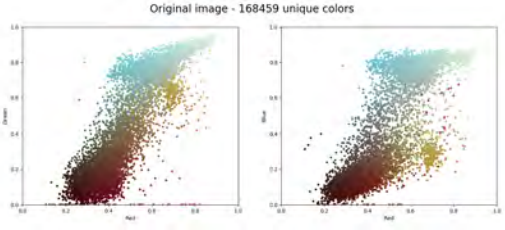
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 168459 unique colors

FIGURE A.18. Sailboat - Various Kmeans color scatter plots (2, 4, 8, 16, 32, 64, 128, 256, & 168459)

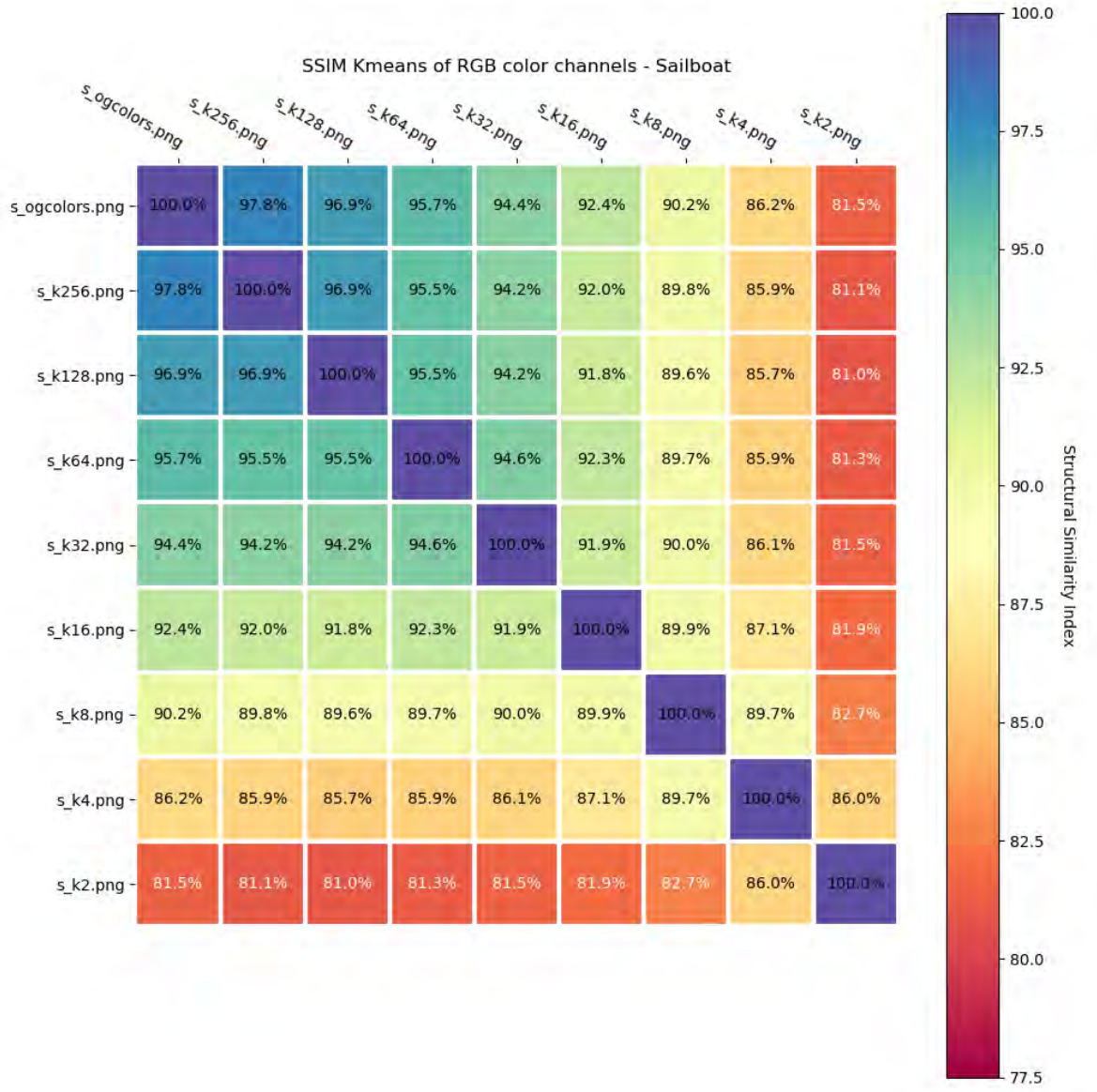


FIGURE A.19. SSIM Heatmap - Kmeans RGB color channels - Sailboat

A.10. Results - Random Color Selection - Sailboat

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

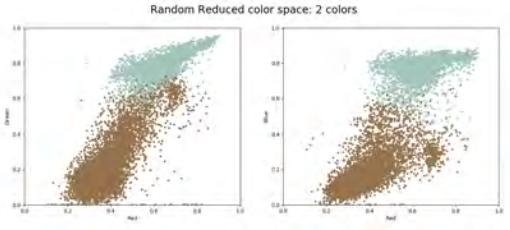
(F) 64-color clusters

(G) 128-color clusters

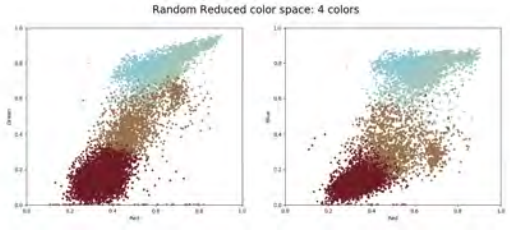
(H) 256-color clusters

(I) Original Image

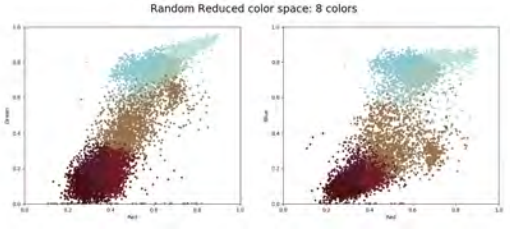
FIGURE A.20. Sailboat - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



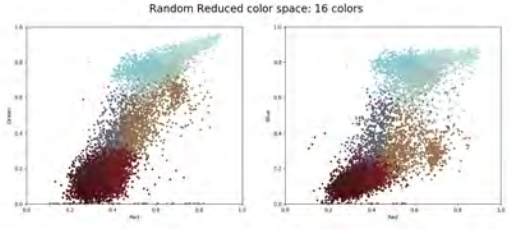
(A) 2 colors



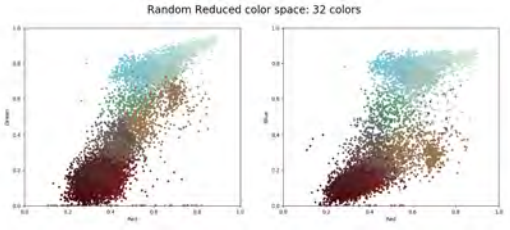
(B) 4 colors



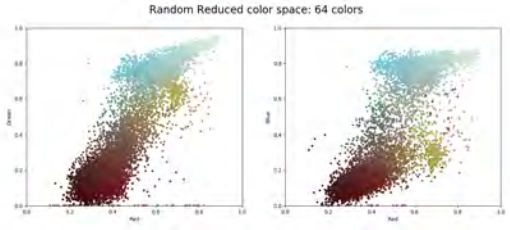
(C) 8 colors



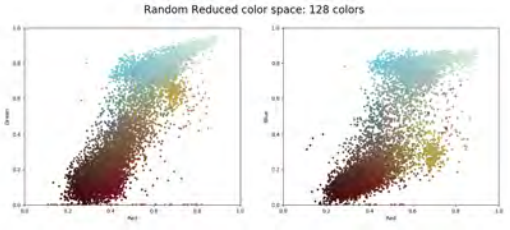
(D) 16 colors



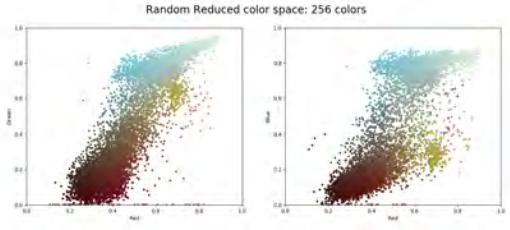
(E) 32 colors



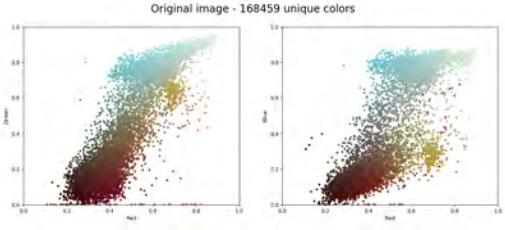
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 168459 unique colors

FIGURE A.21. Sailboat - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & 168459)

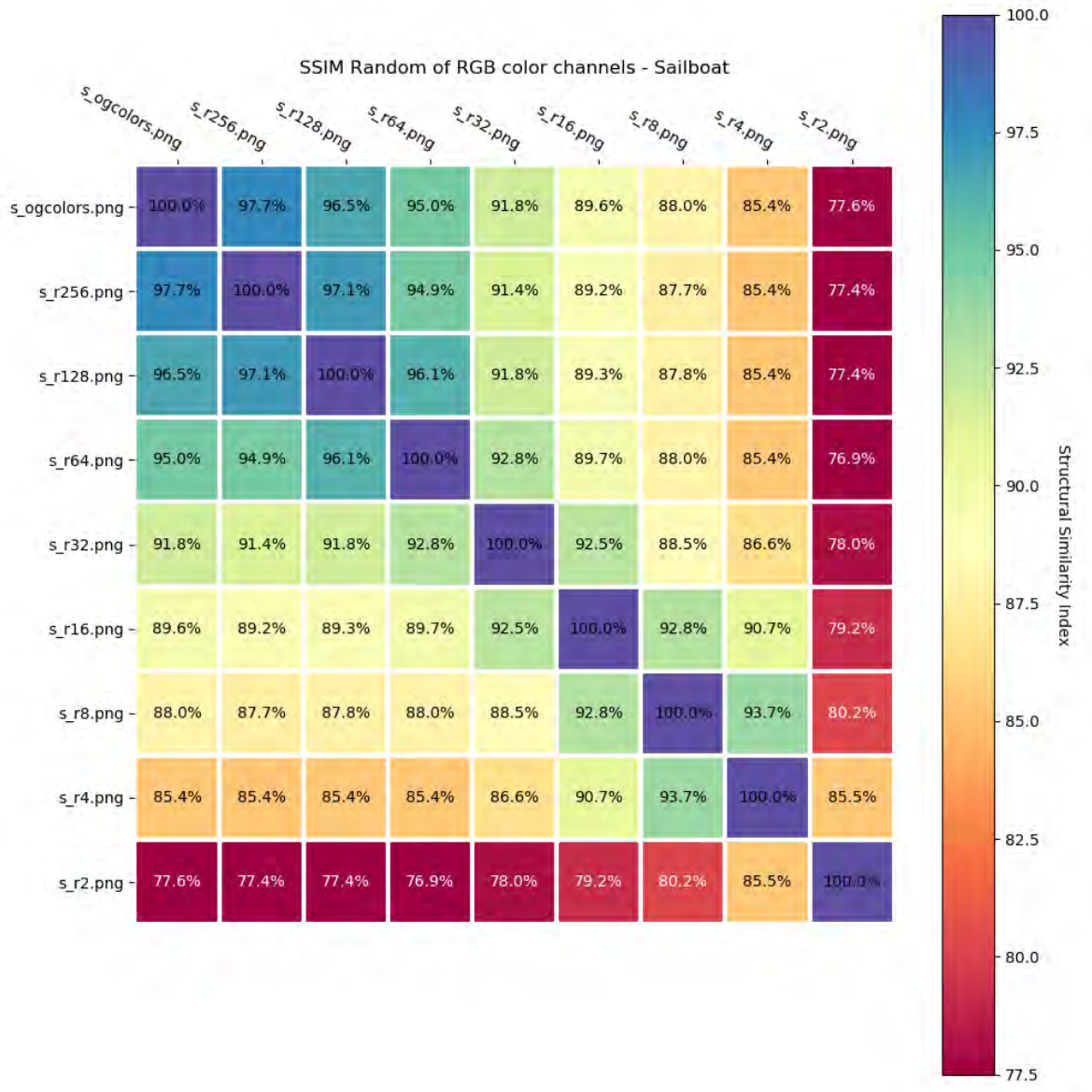
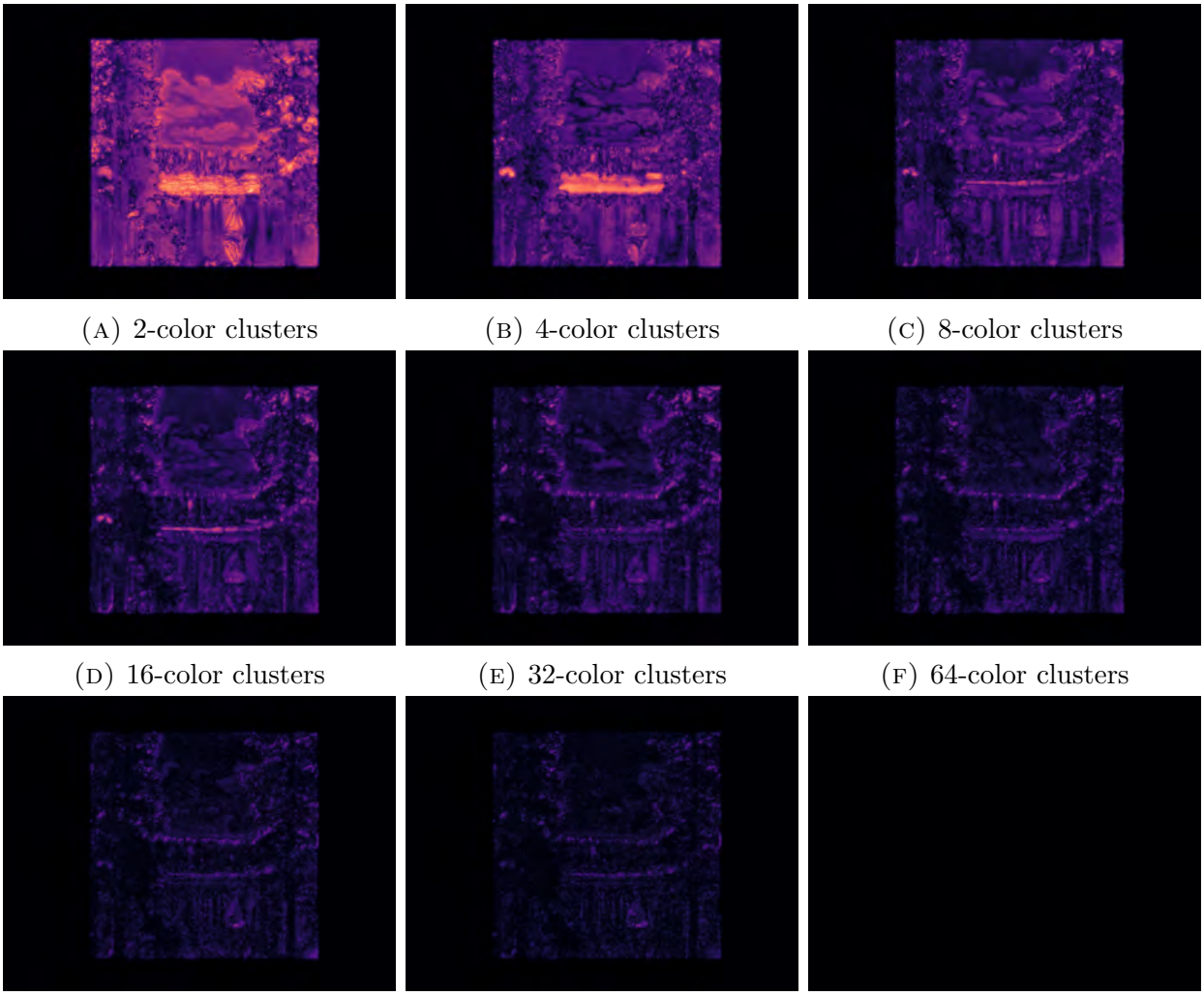


FIGURE A.22. SSIM Heatmap - Random RGB color channels - Sailboat

A.11. ꟻLIP Results - Sailboat



(A) 2-color clusters (B) 4-color clusters (C) 8-color clusters

(D) 16-color clusters (E) 32-color clusters (F) 64-color clusters

(G) 128-color clusters (H) 256-color clusters (I) Original Image

FIGURE A.23. Sailboat - ꟻLIP Kmeans color clusters (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

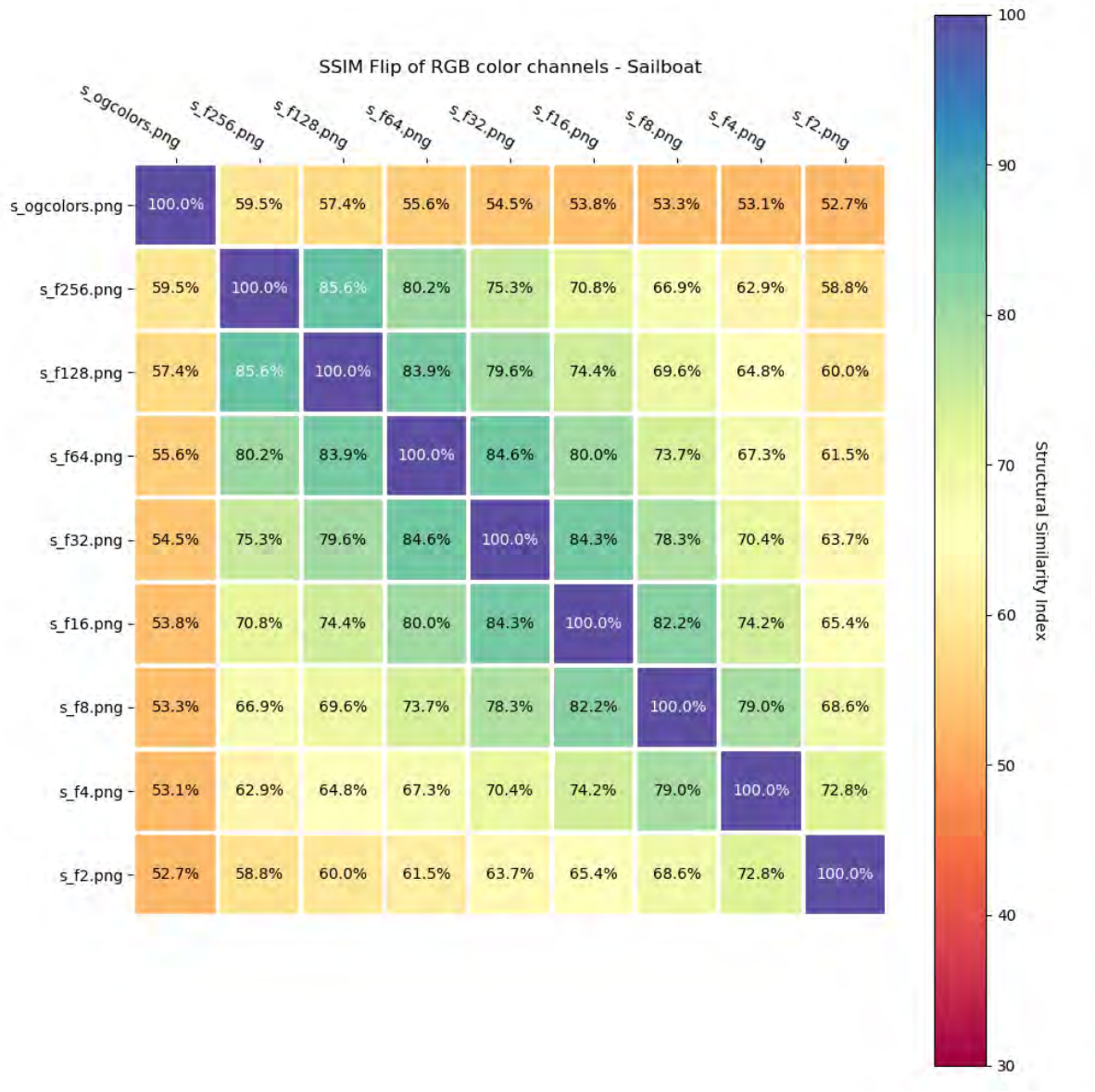


FIGURE A.24. SSIM Heatmap - ꟻLIP Kmeans RGB color channels - Sailboat

A.12. Results - Storage Space - Sailboat

TABLE A.3. Sailboat Compression Comparison

	Storage Size	
Cluster Size	Kmeans	Random Selection
Original	347.2 kB	347.2 kB
256	316.8 kB	327.9 kB
128	285.9 kB	297.7 kB
64	245.6 kB	244.1 kB
32	179.8 kB	162.5 kB
16	115.5 kB	110.0 kB
8	64.8 kB	66.6 kB
4	28.5 kB	31.8 kB
2	13.7 kB	11.3 kB

A.13. Results - Kmeans - Tree

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

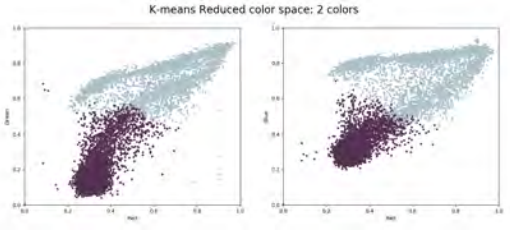
(F) 64-color clusters

(G) 128-color clusters

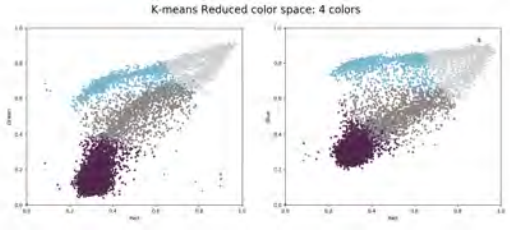
(H) 256-color clusters

(I) Original Image

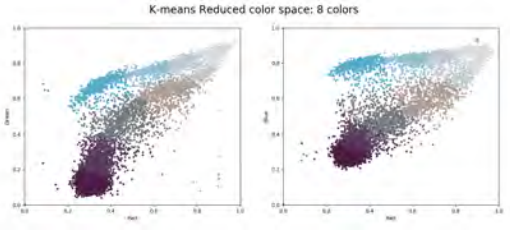
FIGURE A.25. Tree - Various Kmeans color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



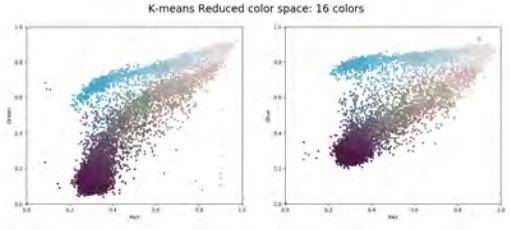
(A) 2 colors



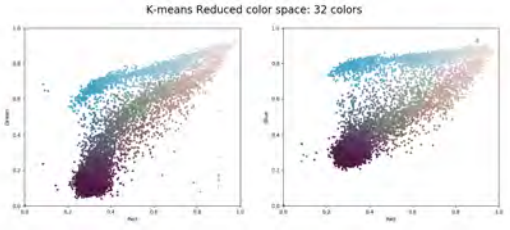
(B) 4 colors



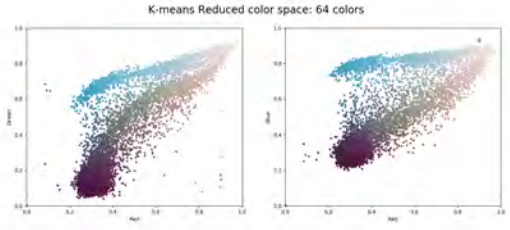
(C) 8 colors



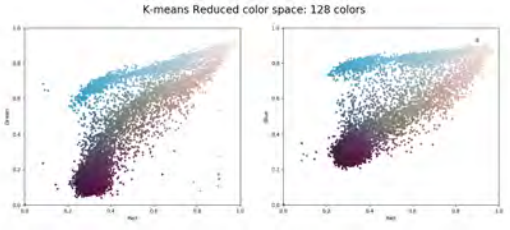
(D) 16 colors



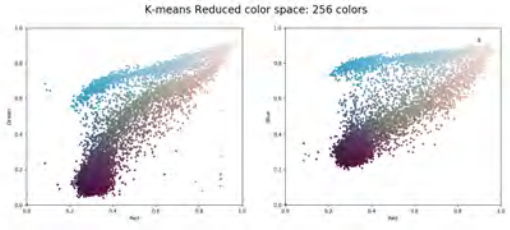
(E) 32 colors



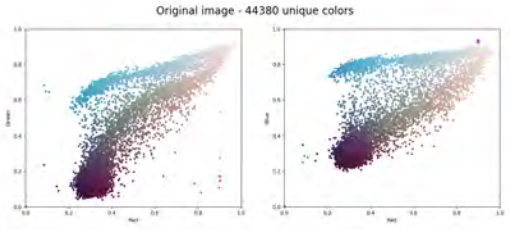
(F) 64 colors



(G) 128 colors



(H) 256 colors



(I) 44380 unique colors

FIGURE A.26. Tree - Various Kmeans color scatter plots (2, 4, 8, 16, 32, 64, 128, 256, & 44380)

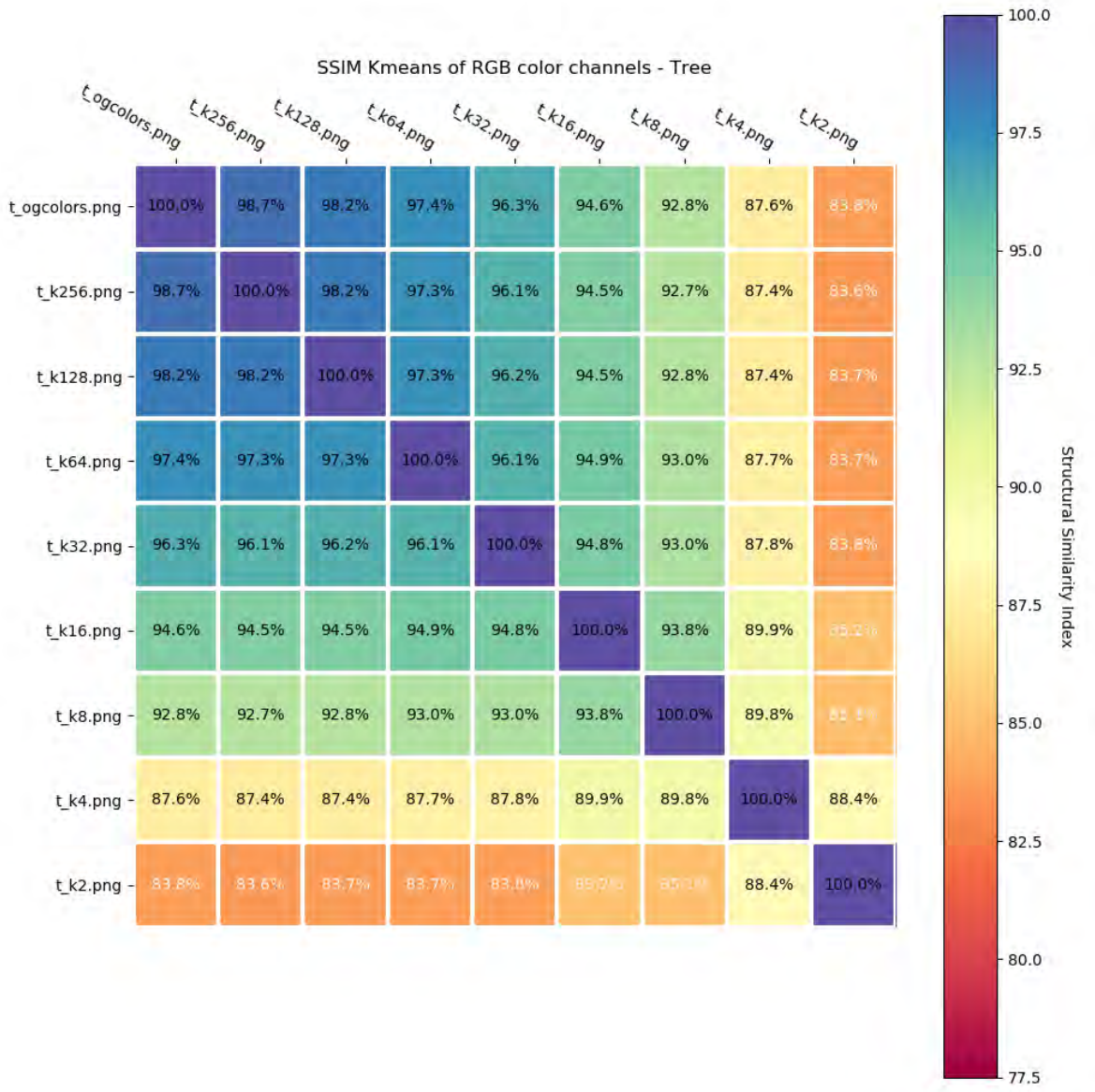


FIGURE A.27. SSIM Heatmap - Kmeans RGB color channels - Tree

(A) 2-color clusters

(B) 4-color clusters

(C) 8-color clusters

(D) 16-color clusters

(E) 32-color clusters

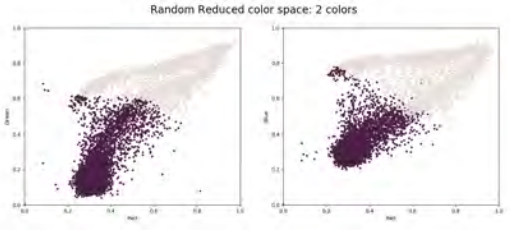
(F) 64-color clusters

(G) 128-color clusters

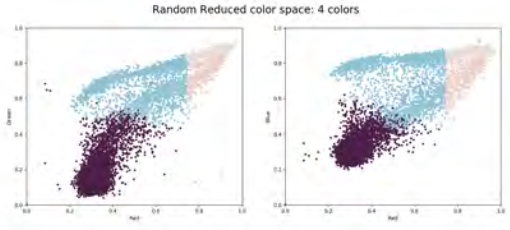
(H) 256-color clusters

(I) Original Image

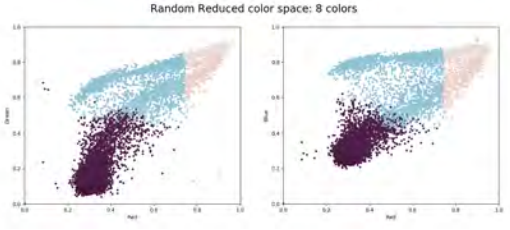
FIGURE A.28. Tree - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)



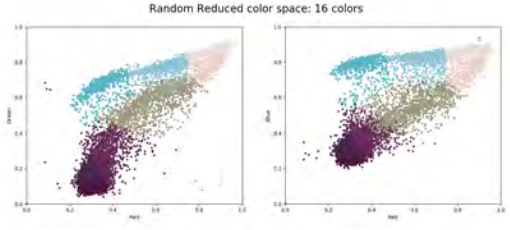
(A) 2 colors



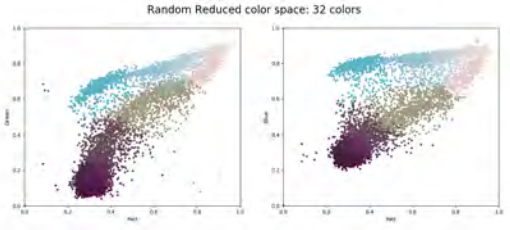
(B) 4 colors



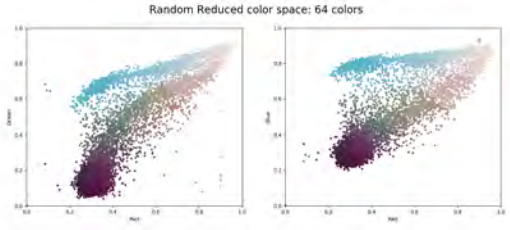
(C) 8 colors



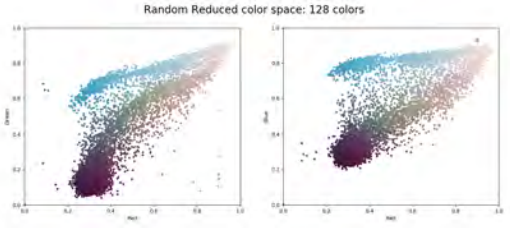
(D) 16 colors



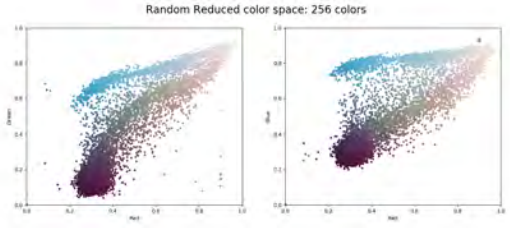
(E) 32 colors



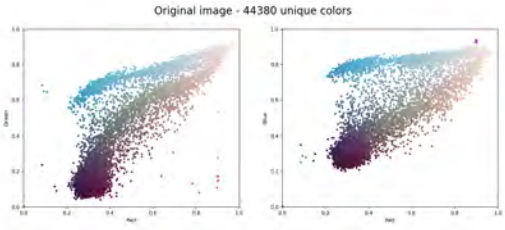
(F) 64 colors



(G) 128 colors



(H) 256colors



(I) 44380 unique colors

FIGURE A.29. Tree - Various Random color clusters applied (2, 4, 8, 16, 32, 64, 128, 256, & 44380)

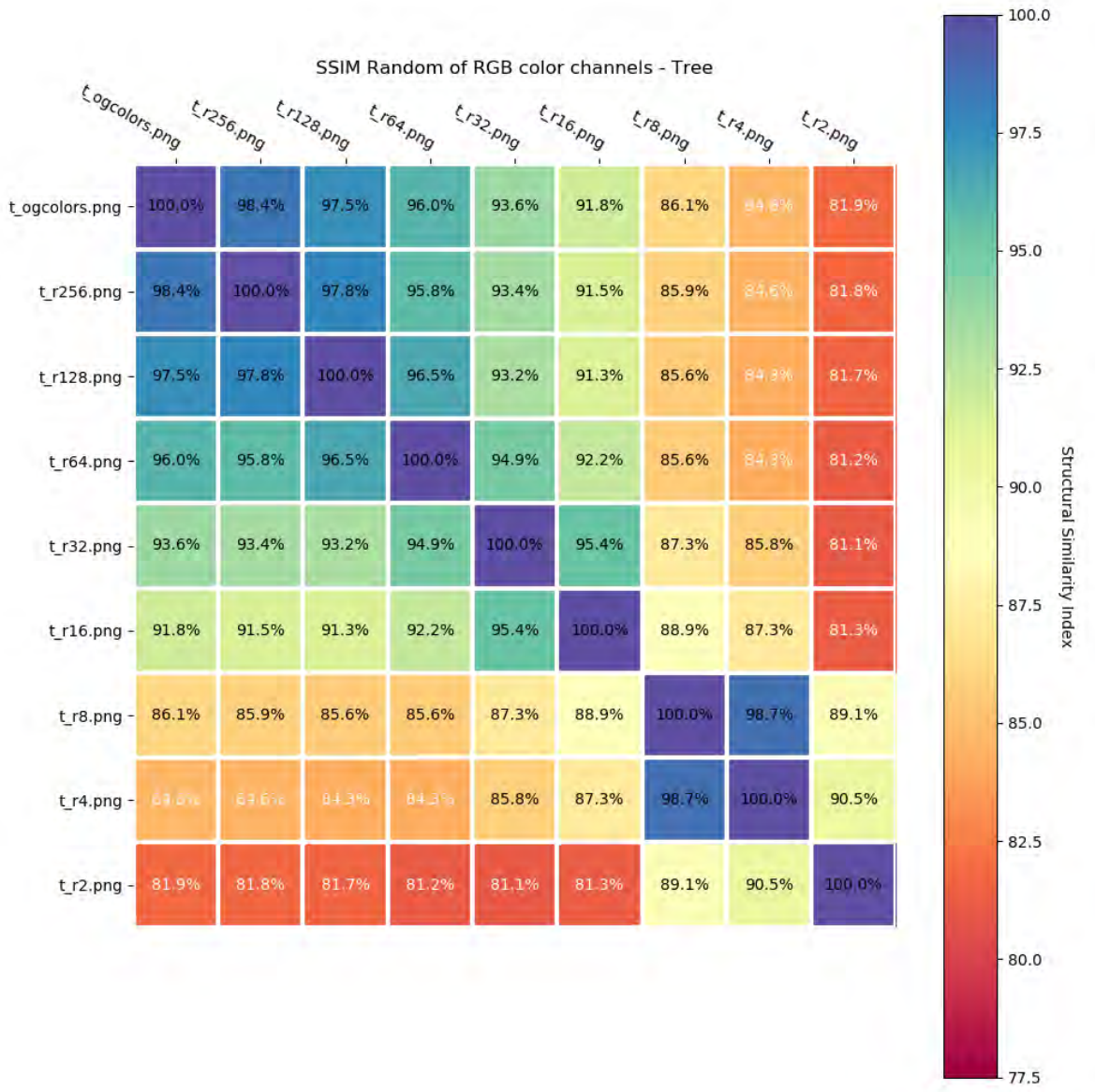
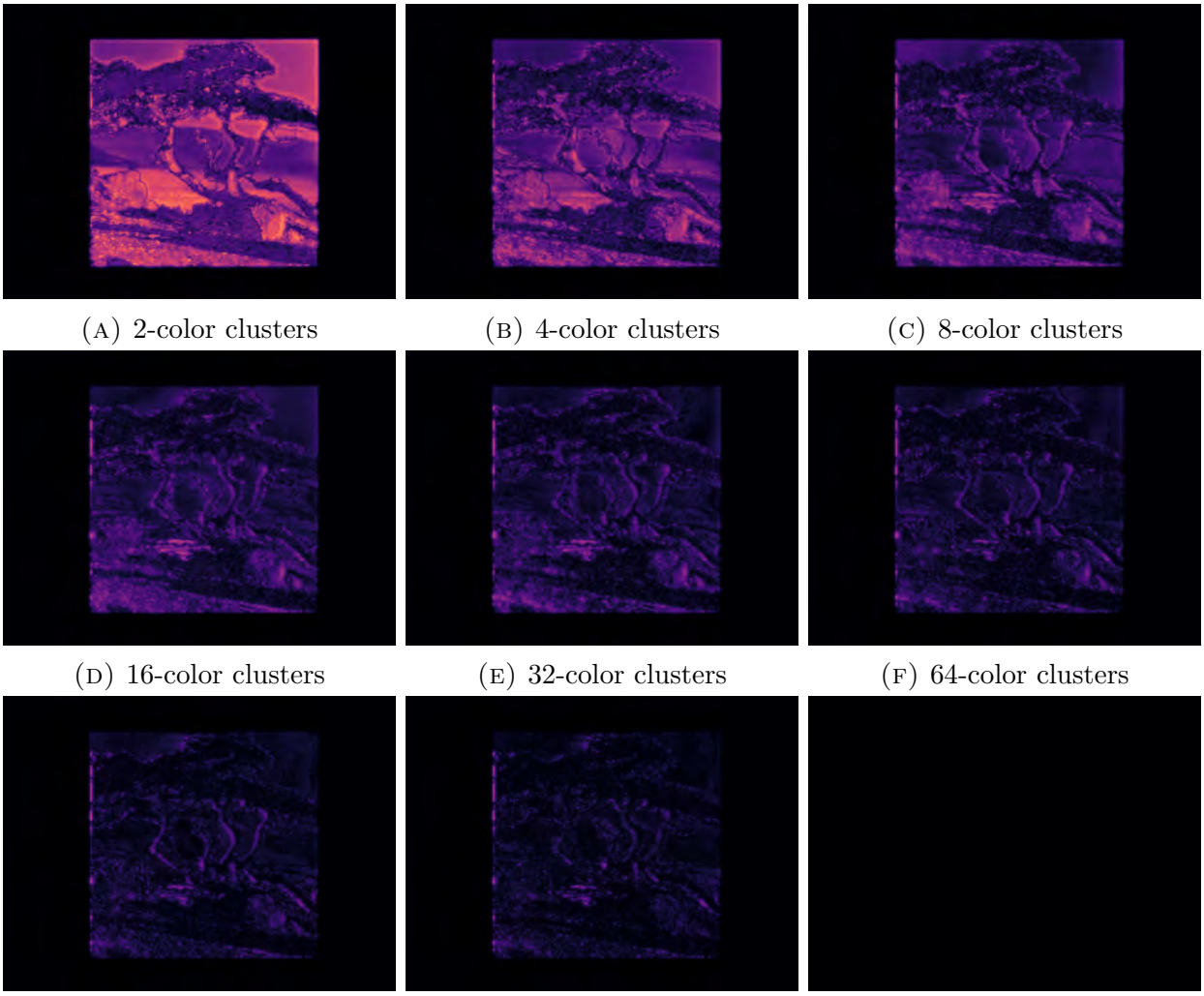


FIGURE A.30. SSIM Heatmap - Random RGB color channels - Tree

A.15. ꟻLIP Results - Tree



(A) 2-color clusters (B) 4-color clusters (C) 8-color clusters

(D) 16-color clusters (E) 32-color clusters (F) 64-color clusters

(G) 128-color clusters (H) 256-color clusters (I) Original Image

FIGURE A.31. Airplane - ꟻLIP Kmeans color clusters (2, 4, 8, 16, 32, 64, 128, 256, & Original Image)

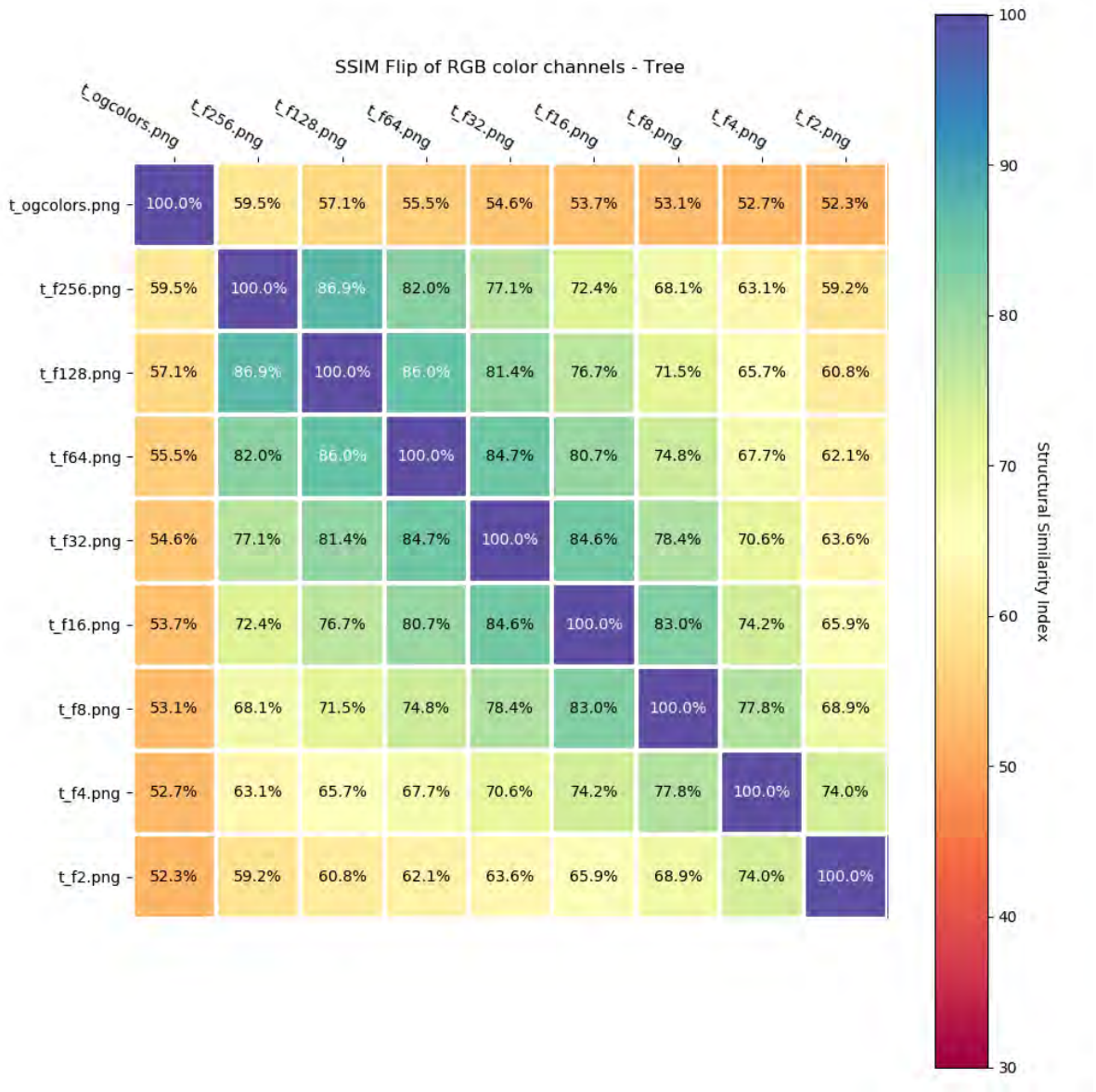


FIGURE A.32. SSIM Heatmap - ꟻLIP Kmeans RGB color channels - Tree

TABLE A.4. Tree Compression Comparison

	Storage Size	
Cluster Size	Kmeans	Random Selection
Original	189.3 kB	189.3 kB
256	159.6 kB	165.1 kB
128	141.5 kB	145.1 kB
64	117.1 kB	116.1 kB
32	87.2 kB	76.0 kB
16	51.7 kB	56.1 kB
8	33.8 kB	25.0 kB
4	17.8 kB	16.1 kB
2	9.9 kB	10.0 kB

REFERENCES

[1] AAMI EC57:2012, *Testing and reporting performance results of cardiac rhythm and ST segment measurement algorithms*, 2012.

[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[3] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[4] M. Fatica, "Cuda toolkit and libraries," in *2008 IEEE Hot Chips 20 Symposium (HCS)*, Aug 2008, pp. 1–22.

[5] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[7] J. T. Behrens, "Principles and procedures of exploratory data analysis." *Psychological Methods*, vol. 2, no. 2, p. 131, 1997.

[8] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[9] T. Parr and J. Howard, "The matrix calculus you need for deep learning," *CoRR*, vol. abs/1802.01528, 2018. [Online]. Available: http://arxiv.org/abs/1802.01528

[10] H. J. KELLEY, "Gradient theory of optimal flight paths," *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960. [Online]. Available: https://doi.org/10.2514/8.5282

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[12] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Annals of Mathematical Statistics*, vol. 23, pp. 462–466, 1952.

[13] H. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 2007.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[15] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[17] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[18] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[19] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," 1967.

[20] G. Gan, C. Ma, and J. Wu, *Data clustering: theory, algorithms, and applications*. SIAM, 2020.

[21] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image segmentation using k-means clustering algorithm and subtractive clustering algorithm," *Procedia Computer Science*, vol. 54, pp. 764–771, 2015.

[22] Y. Yasami and S. P. Mozaffari, "A novel unsupervised classification approach for network anomaly detection by k-means clustering and id3 decision tree learning methods," *The Journal of Supercomputing*, vol. 53, no. 1, pp. 231–245, 2010.

[23] W. Zhong, G. Altun, R. Harrison, P. C. Tai, and Y. Pan, "Improved k-means clustering algorithm for exploring local protein sequence motifs representing common structural property," *IEEE transactions on Nanobioscience*, vol. 4, no. 3, pp. 255–265, 2005.

[24] University of Southern California - Department of Electrical Engineering. (1977) The USC-SIPI Image Database. [Online]. Available: http://sipi.usc.edu/database/database.php

[25] M. E. Celebi, "Fast color quantization using weighted sort-means clustering," *J. Opt. Soc. Am. A*, vol. 26, no. 11, pp. 2434–2443, Nov 2009. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-26-11-2434

[26] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, "ꟻLIP: A Difference Evaluator for Alternating Images," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 3, no. 2, pp. 15:1–15:23, 2020.

[27] J. A. Bilmes *et al.*, "A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," *International Computer Science Institute*, vol. 4, no. 510, p. 126, 1998.

[28] P. Grother, "Nist special database 19. nist handprinted forms and characters database," March 16, 1995.

[29] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[30] Centers for Disease Control and Prevention, "Heart disease facts," "CDC.gov https://www.cdc.gov/heartdisease/facts.htm".

[31] ——, "Underlying cause of death, 1999-2019," "CDC.gov https://wonder.cdc.gov/controller/datarequest/D76".

[32] A. M. Rateb, "A fast compressed sensing decoding technique for remote ecg monitoring systems," *IEEE Access*, vol. 8, pp. 197 124–197 133, 2020.

[33] T. Lili and H. Wei, "Portable ecg monitoring system design," in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, Oct 2019, pp. 1370–1373.

[34] E. Brophy, W. Muehlhausen, A. F. Smeaton, and T. E. Ward, "Cnns for heart rate estimation and human activity recognition in wrist worn sensing applications," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2020, pp. 1–6.

[35] X. Zhang and Y. Lian, "A 300-mv 220-nw event-driven adc with real-time qrs detection for wearable ecg sensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 6, pp. 834–843, Dec 2014.

[36] M. Sharma, R. Tan, and U. Acharya, "Automated heartbeat classification and detection of arrhythmia using optimal orthogonal wavelet filters," *Informatics in Medicine Unlocked*, vol. 16, p. 100221, 2019.

[37] T. Li and M. Zhou, "Ecg classification using wavelet packet entropy and random forests," *Entropy*, vol. 18, no. 8, 2016. [Online]. Available: https://www.mdpi.com/1099-4300/18/8/285

[38] M. Kachuee, S. Fazeli, and M. Sarrafzadeh, "Ecg heartbeat classification: A deep transferable representation," *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, Jun 2018. [Online]. Available: http://dx.doi.org/10.1109/ICHI.2018.00092

[39] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *IEEE Engineering in Medicine and Biology Magazine*, vol. 20, no. 3, pp. 45–50, 2001.